Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2018

David Moisés Pacheco de Araújo e Silva

Interface Gráfica para um Sistema de Gestão de Eventos da Vida

Graphical Interface for a Life Event Management System

Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2018

David Moisés Pacheco de Araújo e Silva

Interface Gráfica para um Sistema de Gestão de Eventos da Vida

Graphical Interface for a Life Event Management System

"In three words I can sum up everything
I've learned about life: it goes on."

— Robert Frost

David Moisés Pacheco de Araújo e Silva

Interface Gráfica para um Sistema de Gestão de Eventos da Vida

Graphical Interface for a Life Event Management System

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Prof. Doutor Hélder José Rodrigues Gomes, Professor Adjunto na Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro e do Prof. Doutor André Ventura da Cruz Marnoto Zúquete, Professor Auxiliar no Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Prof. Doutor Joaquim João Estrela Ribeiro Silvestre Madeira
Professor Auxiliar no Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Carlos Alberto Pacheco dos Anjos Duarte
Professor Auxiliar na Faculdade de Ciências da Universidade de Lisboa (Arguente Principal)

Prof. Doutor Hélder José Rodrigues Gomes
Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

É com muito orgulho que agradeço aos meus pais por todos os sacrifícios por que passaram para me darem tudo aquilo que precisei para que conseguisse chegar até aqui. Nunca vos conseguirei retribuir aquilo que me proporcionaram.

Igualmente agradeço a toda a minha família pelo apoio que me deram e pela insistência em me motivarem para este trabalho. Em especial, quero agradecer ao Mário e à Clara, que sempre estiveram disponíveis e foram impecáveis em tudo o que precisei durante o meu percurso académico.

Quero também deixar um agradecimento especial aos meus orientadores, o Professor Hélder Gomes e o Professor André Zúquete, pela paciência que tiveram durante o período de realização deste trabalho e por todo o apoio que deram à realização do mesmo.

Agradeço também à Professora Beatriz Sousa Santos por toda a documentação que me facilitou para que este trabalho fosse possível.

Quero também agradecer ao Ricardo e a todos os meus amigos que me acompanharam durante o meu percurso académico e com quem cresci enquanto pessoa entre debates e noitadas de estudo. Cada um de vocês merece mais do que um agradecimento genérico, mas se fizesse uma dedicatória individual a cada um, a minha tese teria mais agradecimentos do que conteúdo.

Por fim, quero agradecer à Universidade de Aveiro pela excelente experiência que tive enquanto aluno, não só a nível de ensino como também pelas pessoas que aqui conheci e que me ajudaram a crescer tanto a nível pessoal como profissionalmente.

palavras-chave interface gráfica, usabilidade, serviços orientados a eventos da vida, chappie

resumo O modelo de prestação de serviços orientados a eventos da vida (OEV) tem como objectivo prestar serviços ao cidadão directamente associados a episódios específicos do dia-a-dia destes, o que pode envolver vários organismos da Administração Pública (AP). Estes serviços têm sido implementados com base na comunicação direta entre os organismos da AP envolvidos e isto pode ser questionável do ponto de vista da privacidade do cidadão. O modelo CHAPAS (Citizen HAndled Public Administration Services) de prestação de serviços OEV, foi proposto com o objetivo de desincentivar a comunicação direta entre instituições para a obtenção de informação do cidadão necessária à prestação de um serviço OEV, colocar o cidadão no controlo da disseminação da sua informação entre as várias instituições e fomentar a minimização da informação que o cidadão tem de fornecer às várias instituições para obter os serviços que pretende. Para isso, o modelo pressupõe o uso de uma aplicação, o Chappie, para fazer a composição dos requisitos de cada serviço e auxiliar o cidadão no processo da prestação do mesmo.

Esta aplicação, contudo, tinha uma interface de utilizador extremamente rudimentar que precisava de ser melhorada. É nesse contexto que surge este trabalho, que tem como fim dotar o Chappie de uma interface gráfica moderna e capaz de lidar com a complexidade inerente à integração vários serviços, eventualmente prestados por múltiplos organismos da AP. Para isso, foram desenhados protótipos e conduzidos testes de usabilidade sobre estes no sentido de perceber qual o modelo de interação com o cidadão mais adequado.

O resultado final deste trabalho é uma aplicação baseada em tecnologias Web capaz de ser executada em computadores pessoais e tablets que, usando uma vista tradicional de assistente, auxilia o cidadão nos passos necessários para a obtenção dos serviços de acordo com o modelo CHAPAS.

keywords user interface, usability, life event services, chappie

abstract

The Life-Event Service (LES) provision model has the objective of providing services to citizens related to specific episodes of their daily lives that may involve several Public Administration (PA) organisms. These services have traditionally been implemented based on direct communication between organisms of the Public Administration and that might be questionable from a citizen's privacy standpoint. The Citizen-side HAndling of Public Administration e-Services (CHAPAS) model of LES provision was proposed with the objective of discouraging direct communication between the institutions involved in a LES for obtaining the required personal information from the citizen, put the citizen in control of the dissemination of his personal information through the several institutions and promote the minimization of the information that a citizen has to supply to an institution for requesting a service that he is in the need of. To achieve this, the model requires the use of an application, Chappie, that is responsible for fulfilling the requirements for each of the services involved in the process and assist the citizen to obtain it.

This application, however, had an extremely rudimentary User Interface that was in the need for several improvements. It is in this context that this work arises, with the goal of endowing Chappie with a modern User Interface capable of handling the complexity that is inherent to the integration of several services, possibly provided by different PA organisms. To achieve this, prototypes were sketched and usability studies on those were conducted in order to understand the more suitable user interaction method.

The end result of this work is an application made with web technologies that can be run in personal computers and tablets that, through the use of a traditional wizard view, assists a citizen in the required steps for obtaining a service using the CHAPAS model.

Contents

Contents i

List of Figures v

List of Tables ix

Glossary xi

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Problem . . . 2
- 1.3 Contribution . . . 3
- 1.4 Dissertation Structure . . . 3

2 State of the art and related work 5
- 2.1 Service provisioning in E-Government . . . 5
  - 2.1.1 Provision of disperse services . . . 5
  - 2.1.2 Provision of integrated services . . . 6
  - 2.1.3 Mobile Government (M-Government) . . . 7
- 2.2 Life Event Services (LES) . . . 8
  - 2.2.1 Life Event Portals . . . 8
  - 2.2.2 CHAPAS Model . . . 9
- 2.3 User Interaction Models . . . 10
  - 2.3.1 Wizards . . . 10
    - Navigation guides . . . 11
  - 2.3.2 Conversational interfaces . . . 11
    - Chatbots . . . 12
    - Virtual Personal Assistants . . . 12
- 2.4 Reference Implementations . . . 13
  - 2.4.1 Oracle Peoplesoft eBenefits . . . 13
  - 2.4.2 Portal do Cidadão (Citizen Portal) . . . 15
  - 2.4.3 Paredes Municipality e-Services portal . . . 17
- 2.5 Usability Evaluation . . . 21
  - 2.5.1 Usability Standards . . . 21
  - 2.5.2 Heuristic Evaluation . . . 22
  - 2.5.3 Cognitive Walkthrough . . . 24
  - 2.5.4 Usability Tests . . . 24

3 Chappie **25**
3.1 Building the Service Dependencies Tree . . . 25
3.1.1 User Interaction During The Tree Building Process . . . 25
3.2 Obtaining Services in Chappie . . . 28
3.2.1 User interaction during the process . . . 28
3.3 Usability Problems . . . 28
3.4 Technical Challenges . . . 31

4 Prototyping the User Interface **35**
4.1 User profile . . . 35
4.2 Expected Features . . . 35
4.2.1 Home Screen . . . 36
4.2.2 Citizen Wallet . . . 36
4.2.3 History . . . 38
4.2.4 Use of a wizard view to guide the users . . . 38
4.2.5 Automatic information input . . . 38
4.3 Usability goals . . . 39
4.4 System requirements . . . 39
4.5 Platform limitations . . . 40
4.6 First usability test . . . 40
4.6.1 Participants . . . 41
4.6.2 General instructions . . . 41
4.6.3 Task 1: Add a personal detail (Taxpayer number) . . . 42
Expected steps . . . 42
Results from the usability test . . . 44
4.6.4 Task 2: Begin a service and obtain its required documents . . . 45
Expected steps . . . 45
Results from the usability test . . . 48
4.6.5 Task 3: Fill-in input fields . . . 50
Expected steps . . . 50
Results from the usability test . . . 54
4.6.6 Task 4: Understand that the service was finished . . . 55
Expected stepts . . . 55
Results from the usability test . . . 55
4.6.7 Task 5: Switch Sectors . . . 55
Expected stepts . . . 55
Results from the usability test . . . 57
4.6.8 Conclusions . . . 57
4.7 Second paper prototype . . . 58
4.7.1 Participants . . . 58
4.7.2 General instructions . . . 59
4.7.3 Task 1: Add a personal detail (Taxpayer number) . . . 59
Expected steps . . . 59
4.7.4 Usability test's results . . . 61
4.7.5 Task 2: Find the required service . . . 66
Expected steps . . . 66
4.7.6 Usability test's results . . . 66

4.7.7 Task 3: Fill-in the information requested by the service 70
Expected steps 71
4.7.8 Usability test's results 71
4.7.9 Task 4: Obtain the service 75
Expected steps 76
4.7.10 Usability test's results 76
4.7.11 Task 5: View obtained documents 78
Expected steps 78
4.7.12 Usability test's results 80
4.7.13 Overall appreciation 80

5 Developing the new interface **83**
5.1 Usage of Web Technology for the UI 83
5.2 Chappie API 85
5.3 Home screen 86
5.4 Initial approach to dependencies tree building 86
5.4.1 Feedback from the users 93
5.5 Adopted approach for obtaining services 93
5.6 Technical details of the implemented solution 99

6 Conclusion **101**
6.1 Considerations on the work done 101
6.2 Future work 102

List of References **103**

List of Figures

2.1 Operating System Market Share Worldwide between 2010 and 2018 7
2.2 Sequence of interactions for obtaining services in Chappie (source [9]) 10
2.3 Oracle PeopleSoft: Select Life Event Screen (source: [27]) 14
2.4 Oracle PeopleSoft: Task Status (source: [27]) 14
2.5 Oracle PeopleSoft: Personal Information Sub-task (source: [27]) 15
2.6 Oracle PeopleSoft: Complete Event and Exit Screen (source: [27]) 16
2.7 Excerpt from a service page in Portal do Cidadão 16
2.8 Documents bag in Portal do Cidadão 17
2.9 Excerpt from the Forms Page in Paredes Municipality's E-Services Portal . . . 18
2.10 Excerpt from a service page in Paredes Municipality's E-Services Portal 19

3.1 Process Chappie uses to create a dependencies tree 26
3.2 Illustrative example of a dependencies tree for an hypothetical service named 1A. 26
3.3 Process for obtaining services in the CHAPAS Model 27
3.4 Circumstance input in the old Chappie prototype. 29
3.5 Provider selection in the old Chappie prototype. 30
3.6 Data input screen in the old Chappie prototype. 30
3.7 An obtained service in the old Chappie prototype. 31
3.8 Workflow followed by the old Chappie prototype to create a dependencies tree and list all the required documents. 33

4.1 Early concept of the Chappie home screen. 36
4.2 Early concept of a screen where the citizen could select to reuse a document from the wallet. 37
4.3 Early concept of screen for selecting documents from the wallet. 37
4.4 Chappie initial screen in the first paper prototype 42
4.5 "People" screen in the first paper prototype 43
4.6 New Item dialogue in the first paper prototype 43
4.7 Screen showing the list of services available in the "Encontro Nacional de Estudantes" sector. 46
4.8 Chappie screen to start the National Students Encounter sign-up service. 46
4.9 A circumstance in the first paper prototype. 47
4.10 Error message shown if the user is not a student 47
4.11 Enumeration of required documents for the service 48
4.12 Screen showing the possibility of document reusage 49
4.13 Select provider screen of the first paper prototype 49

4.14 Message shown after a document being obtained . . . 50
4.15 Options shown to the user when inputing a field. . . . 52
4.16 Menu for selecting a field value from a folder . . . 52
4.17 Available auto-fill items presented in the "Eu" (Me) folder . . . 53
4.18 Auto-fill screen with keyboard accelerators displayed on the right . . . 53
4.19 Review service screen . . . 56
4.20 Success message displayed when the user finishes the service . . . 56
4.21 Change sectors drop-down . . . 57
4.22 Chappie on-boarding welcome screen . . . 60
4.23 Create Profile Screen . . . 60
4.24 User Profile Form . . . 61
4.25 Add identity cards screen . . . 62
4.26 Manual insertion of field values in an Identity Card. . . . 62
4.27 Chappie importing data from a smartcard . . . 63
4.28 Adding a custom field to an Identity Card. . . . 63
4.29 Indicator used in an already obtained card. . . . 64
4.30 On-boarding finish screen. . . . 64
4.31 Chappie log-in screen . . . 66
4.32 Chappie PIN input screen . . . 67
4.33 Chappie search screen . . . 67
4.34 Chappie search results screen . . . 68
4.35 List of services shown in the second prototype . . . 68
4.36 List of services available in the "Documents" category . . . 69
4.37 Event description page . . . 69
4.38 Welcome screen for a service in second paper prototype . . . 71
4.39 Provider selection screen in second paper prototype . . . 72
4.40 Required partial services for obtaining a document . . . 72
4.41 Welcome screen for a partial service in second paper prototype . . . 73
4.42 Circumstance of a partial service in the second paper prototype . . . 73
4.43 Data input required for a partial service in the second paper prototype . . . 74
4.44 Review screen for a service in the second paper prototype . . . 74
4.45 Waiting screen shown while obtaining a document . . . 76
4.46 Additional solicitation required for service conclusion . . . 77
4.47 Service success screen of the second paper prototype . . . 77
4.48 Finding obtained documents in the citizen wallet . . . 79
4.49 Finding obtained documents in the history screen . . . 79

5.1 High-level view of the Apache Cordova application architecture (source: Apache Cordova Documentation). . . . 84
5.2 Workflow followed by the new Chappie application to create a dependencies tree and obtain all the required documents. . . . 87
5.3 The home screen of the final application. . . . 88
5.4 Early concept of a tree view using half of the screen . . . 89
5.5 An example of nodes requesting attention from the user . . . 89
5.6 Service information dialog . . . 90
5.7 Service circumstance dialog . . . 90
5.8 Select provider dialog . . . 91

5.9 List of document attributes 91
5.10 Input form dialog 92
5.11 The tree drawn using the whole screen. 92
5.12 Welcome screen on the final application 94
5.13 Circumstance screen in the final application 95
5.14 Input form in the final application 95
5.15 List of required documents for a partial service 96
5.16 List of document attributes 96
5.17 Document that the citizen must upload 97
5.18 Citizen wallet for selecting the document to upload 98
5.19 Select provider screen in the final application 98
5.20 Review screen in the final application 99
5.21 Success screen in the final application 100

List of Tables

4.1 Details on the participants of the first usability test 41
4.2 Results from the first task of the first usability test 44
4.3 Results from the second task of the first usability test 51
4.4 Results from the third task of the first usability test 54
4.5 Success rate of the fourth and fifth task of the first usability test 58
4.6 Details on the people who participated in the second usability test 59
4.7 Observer's notes for the first task of the second usability test 65
4.8 Reported and observed difficulty of the first task of the second usability test . . 65
4.9 Observer's notes for the second task of the second usability test. 70
4.10 Reported and observed difficulty of the second task of the second usability test. 70
4.11 Observer's notes for the third task of the second usability test. 75
4.12 Reported and observed difficulty of the third task of the second usability test . 75
4.13 Observer's notes for the fourth task of the second usability test. 78
4.14 Reported and observed difficulty of the fourth task of the second usability test. 78
4.15 Reported and observed difficulty of the fifth task of the second usability test. . 80
4.16 Observer's notes for the fifth task of the second usability test. 81
4.17 Observer's notes for the fifth task of the second usability test. 81
4.18 Feedback given by the users after testing the system 82

5.2 List of messages sent from the Chappie Server to the Chappie Client 85
5.1 List of messages sent from Chappie Citizen to Chappie Server 86
5.3 Success rate of the fourth and fifth task of the first usability test 93

Glossary

API Application Programming Interface.

CHAPAS Citizen-side HAndling of Public Administration e-Services.

Chappie Citizen Chappie Citizen Application.

Chappie Server Chappie Server Application.

CSS Cascading Style Sheets.

E-government Electronic Government.

E-Services Electronic Services.

G2B Government to Business.

G2C Government to Citizen.

G2E Government to Employees.

G2G Government to Government.

GUI Graphical User Interface.

HCI Human Computer Interaction.

HHS Guidelines Usability Guidelines of the United States Department of Health and Human Services.

HTML Hypertext Markup Language.

ICT Information and Communication Technologies.

ISO International Organization for Standardization.

Java ME Java Mobile Edition.

Java SE Java Standard Edition.

JFC Java Foundation Classes.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

LES Life-Event Service.

PA Public Administration.

QR Code Quick Response Code.

RDP Required Documents Policy.

REST Representational State Transfer.

SDK Software Development Kit.

UI User Interface.

URL Uniform Resource Locator.

VPA Virtual Personal Assistant.

WAI Web Accessibility Initiative.

Chapter 1

Introduction

The increasing number of people with access to computers, smart-phones, and other Internet connected devices leads to it becoming a more essential way of connecting people and institutions. [1] On-line service usage is growing and leading to simplification and convenience in the way people interact with institutions. [2]

This change in what people expect to be able to do with their devices is being reflected in the type of services that are provided to them. Amongst the institutions that are adapting their service provision models are those that are part of the Public Administration that, through Electronic Government (E-government) programs, are reducing the complexity and the cost of the services they provide. Countries like Malta and Portugal have nearly all of their e-government services automated or fully available on-line. [3]

Despite the benefits people get from the increase in services availability, quality, and automation, this trend may create some privacy risks. For most of these services to work with minimal input from the citizens, the providing institutions must have all the information they need from those. This information can be retrieved from the institution's internal databases or by asking the user or other institutions to provide them. Having an institution querying others for information on a person may raise some privacy concerns. The exchange of information between institutions to provide an unified service may not transparent to the user and sometimes it may not be clear if that information is safely stored or will not be reused in other situations.

In this work it is proposed a Graphical User Interface (GUI) for an application, Chappie, that combines the benefits of an unified service provision without compromising the privacy of the citizens using them.

1.1 Context

The integration of Information and Communication Technologies (ICT) in different services, specifically in Public Administration (PA) Departments, has changed the way institutions interact with citizens. Services once provided by a specific department that only had access to a limited set of personal information of a citizen have become more complex and may now involve the interaction between several departments that need to exchange between them information about that person.

This transition from a discrete service provision model to a transversal service provision (whole-of-government) model where, for the provision of a citizen requested service, an insti-

tution is able to contact other institutions and request additional information (traditionally owned by those institutions) about the said citizen, provides several benefits for both institutions and users, such as less bureaucracy, less time required for performing a task, and overall less costs for both the citizen and the institutions providing the services. [4] [5]

One of these transversal service provisioning models is the LES model. [6] In this model, services are tailored to specific citizens' needs and typically involve a new service that combines several other individual services possibly provided by several different PA departments or institutions. [7] Life-Event Services are typically provided in specific websites called Life-Events portals. In these portals, where the available services are organized according to citizens' needs, a citizen can solve a particular situation in their life through a single Life-event service.

However, the convenience of having a single service to handle all the information may result in a reduced privacy control. In a Life Event Service the citizen may not be aware that he is potentially interacting with different services, which personal information is being exchanged between those services and which are the entities providing them.

Besides, by having access to several portions of a citizen's personal information, it is easier for those institutions to cross data and create unique profiles for the people requesting the service. [8] With the increase in cyber-attacks targeting institutions that own personal identifiable information in the past few years, it is more important than ever to be aware of to whom, when, and what personal information is being transmitted, so that in case something negative happens, the affected people are aware of when to take action.

CHAPAS is a proposal for a Life Event Service interaction model which follows a decentralized architecture where the citizen replaces the role of life-event portals in the handling of the data flows between Public Administration departments. [9]

The CHAPAS model allows provisioning of life event services tailored to the citizen's needs and at the same ensures institutions do not exchange citizen's personal information directly but instead rely on the citizen to handle it to them using an application, Chappie. By doing this, the model ensures the citizens are in control of the disclosure of their personal information.

At its initial state, Chappie required several improvements, namely regarding the user interface as it provided an unpleasant user experience with outdated visual elements. As a way to address these issues, in this work we will propose a set of modifications to this Chappie application. Among those changes is the split of the application in two components: one user application and one server application, which will be described in more detail in the following sections.

1.2 Problem

The CHAPAS Model follows a decentralized architecture where the citizen replaces the role of life-event portals in the handling of the data flows between Public Administration departments [9]. According to this model, services are provisioned by a single entity, output at least one document, and may need the citizen who is using the application to provide them with additional personal information like an address or other documents that must be obtained from other services.

Each service is defined according to a specification, called Required Document Policy (RDP). That specification lists the documents that the service needs, how they can be ob-

tained, and what information the citizen must provide. While it is technically possible to make almost every service available under the CHAPAS Model due to its flexibility and genericness, there are some limitations associated with it that create limitations on the quality of information that is made available to the user in one service that follows this model.

In the CHAPAS Model, services do not contact other services directly when in the need for a specific information or document from a citizen. Instead, services rely on Chappie to handle the flow of documents between institutions. Because Chappie is an application controlled by the citizen, CHAPAS ensures he/she is always in control of what personal information is being requested at each moment and by which entity, creating obstacles on citizen profiling.

As a way to preserve privacy, Chappie is by design agnostic on the type of documents and data that it handles, making it very limited on the extent to which tasks can be automated. This means that while some automation may be possible, it will not prevent the user from having to verify if the information was auto-filled correctly or to input the same information more than once, as Chappie does not know the content of the documents.

The goal of this work was to modify Chappie and adapt it to be a more modern application, suitable for usage in both traditional personal computers as well as in more modern devices such as smart-phones and tablets, while taking into considerations the limitations imposed by the CHAPAS Model that will described later in this document.

1.3 Contribution

The main contribution of this work is the GUI for an application where the provision of a Life-Event Service depends on the discovery and provision of several other partial services. More specifically, the main contribution of this work is a GUI for an already existing application, Chappie, the citizen application of the CHAPAS Life-Event Services Model. We explored several new features that we considered helpful for improving the user experience in Chappie like an home screen from where users can start services, a storage area where users can store personal documents, and the ability of automatic filling fields.

Besides proposing new User Interface features and elements, this work also covers structural modifications to the Chappie application that were made to support the features being proposed, such as allowing the user to define the order that he prefers services to be provisioned and splitting the Chappie application in two: one citizen application written using Web technologies, suitable for both modern devices like desktops and tablets, and a server application that provides the features required by the first using a REST API.

This work also presents the results of two usability tests that were made prior to development and two proposals for building the dependencies tree in Chappie, with one of them being abandoned in the middle of development due to dissatisfaction from the potential users, and the other being the proposed solution for the User Interface of this work.

1.4 Dissertation Structure

This dissertation is organized in 6 chapters. The first Chapter is this one, where the work's context, problem and contribution are presented.

The second chapter covers the definition of E-Government (section 2.1) and Life-Event Services (section 2.2), describes some common user interaction models used in service provisioning (section 2.3) and analyzes applications that were taken as a reference for developing

the User Interface (UI) proposed in this work (section 2.4). This chapter also presents some aspects that have to be considered during the User Interface design process and methods for evaluating the usability of a User Interface (section 2.5).

Chapter 3 describes the initial status of the Chappie application, including the process used for building the dependencies tree (section 3.1), obtaining a service (section 3.2) and asking for user input (section 3.2.1). The chapter also enumerates the usability problems the old prototype had (section 3.3) and the technological challenges that were faced while trying to solve them (section 3.4).

The fourth chapter describes the process that was followed to find a suitable user interface for the application. It describes the profile of the users that Chappie is targeted at (section 4.1), the expected features of the application (section 4.2), the proposed usability goals (section 4.3), the system requirements (section 4.4) and the application limitations (section 4.5). In this chapter the reader can find information on the two usability tests that were conducted and their respective results (sections 4.6 and 4.7).

Chapter 5 describes the process of developing the new application. The chapter covers the technological decisions that were made (section 5.1), and the new API (section 5.2) and home screen (section 5.3).

This chapter also describes the two approaches that were considered for building the dependencies tree and obtaining services: a tree approach, covered in section 5.4, and a wizard approach (the adopted solution) covered in section 5.5). Lastly, the fifth chapter presents the list of libraries used for developing the interface, where the source code can be found and the instructions to build the application (section 5.6).

In the last chapter of this work, Chapter 6, we present a set of considerations based on the work that was done (section 6.1 and describe the future work (section 6.2).

Chapter 2

State of the art and related work

This chapter will describe some of the core concepts used in this work. It starts with an overview of different models of service provisioning in e-government, in section 2.1, specifically service provisioning in a more traditional organizational oriented way (section 2.1.1) and the gradual evolution towards integrated services, explored on section 2.1.2.

We will then briefly explain what Life-Event Services and Life-event portals are in section 2.2 and their importance for the CHAPAS Model and specifically to the Chappie application (section 2.2.2).

This chapter also describes the applications that were used as a reference for developing the Chappie UI (2.4) and the User Interface design guidelines and test methods (section 2.5).

2.1 Service provisioning in E-Government

There is not a formal definition of Electronic Government (e-government). [10] Some authors define e-government as the provisioning of Public Administration Services to the citizen or other entity in need for it, while others consider e-government to be a more broad term that goes beyond service provisioning and includes other areas such as the usage of technologies to promote democracy. [11] Despite those differences, all the different definitions have in common the usage of Information and Communications Technologies to improve communication between governmental agencies and their clients (citizens, companies, and other governmental agencies). [12]

E-governement initiatives are classified in terms of the type of entities to whom services are directed at. Initiatives can be classified as Government to Citizen (G2C), Government to Business (G2B), Government to Employees (G2E), and Government to Government (G2G). [12]

Regarding provisioning models, services can be provided in a disperse manner, where institutions only provide information that is part of their competences, or as part of an integrated provisioning model, where an institution can reach other institutions to gather additional information on a citizen.

2.1.1 Provision of disperse services

Public institutions were initially organized in departmental silos where there was little to no communication with other departments. [13] The first generation of e-government initiatives

were focused mainly on making these services available on-line. [5]

This model, where an institution makes available the services that are strictly part of its competences, is called a disperse service provision model. Disperse because in this government-centric approach services are designed to solve an institution's requirements and are not centered around the needs of the citizens. [14] [15] [16]

For fulfilling a given situation in a person's life, typically several documents are needed that, most of the times, are issued by more than one institution. Take the example of buying a house: in a situation like this, documents issued by Municipalities, Public Administration Departments and even private entities such as banks are required. This implies that the citizen must have some knowledge on how Public Administration services are organized in order to obtain all the documents required to apply for the wanted service.

Due to its complexity and the costs associated with it, this provisioning model negatively affects both the institutions providing the service and the citizen who use them. [17] [18] However, because the institution that provide the service only have access to a limited set of the person's personal information and there is no direct communication with other institutions, it is more difficult to create a citizen profile when one is applying to a service and there is more control of what, how, when, and by whom personal information is being requested. [19]

However, because each institution typically issues a standardized document for every situation, that document may contain more information than what is strictly necessary for the entity that is requesting it to provide its service. [9] As a clarification, let's take an example of a service that even though is not a Public Administration service, is a good example to elucidate the reader about what was said in the previous sentence: when applying for a bank loan one of the required documents is the person's declaration of IRS. That declaration may only be needed to check the annual income of the household but, due to it being a standardized document that can be used to different situations, it also exposes the names and taxpayers number of all the people who are part of that household, providing the bank more information than the strictly necessary to approve or reject the application.

A less bureaucratic approach to service provisioning is the provision of integrated services. In this model institutions not only provide services that are part of their competences but also have the ability to contact other institutions to retrieve additional required information on the citizen that is requesting the service.

2.1.2 Provision of integrated services

As said before, rarely situations a citizen faces in his daily life that require interaction with Public Services are satisfied by obtaining a single document. Typically institutions that issue documents require the citizen to provide them other documents prior to produce the document the citizen is in need.

With the evolution of ICTs and the increased use of e-government services by people, companies, and governmental institutions lead to changes in how services in e-government are provided.

The second generation of e-government initiatives was more focused in a whole-of-government concept or joining-up government. [4] [20] In these models, the institutions communicate and collaborate for aiming at solving a problem in a citizen-centric approach. [14]

Services that are provided by institutions using these models can contact services from other institutions or departments without the citizens having to do them themselves. This allows services to be provision in more modern ways, with a decrease in bureaucracy that is

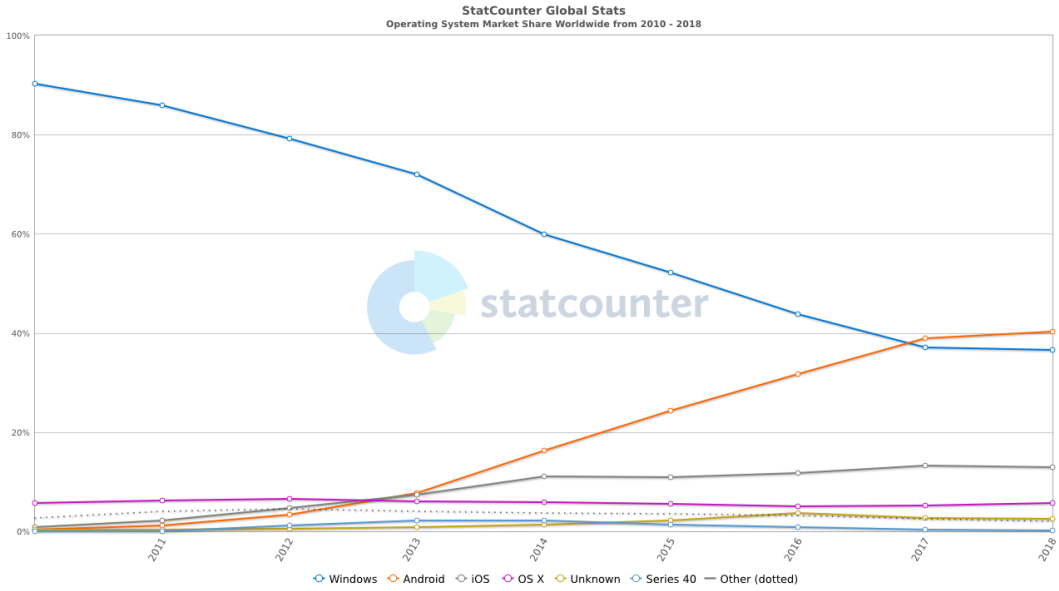

Figure 2.1: Operating System Market Share Worldwide between 2010 and 2018

both beneficial to people and institutions and that leads to an increase satisfaction with public services. This, however, may raise some privacy concerns the previous model did not have as the process of information exchange between institutions is not transparent to the citizen.

2.1.3 Mobile Government (M-Government)

The increasing number of people with devices with smaller screens and with no keyboard and mouse input, like Android and iOS smart-phones and tablets, has made several businesses having to adapt their service provisioning utilities and websites from a more traditional desktop approach to a new mobile-centric approach, where user interfaces have to be adapted for smaller screens and touch inputs. This is happening because Mobile Devices are becoming significantly more important, having its market been on the rise in recent years, according to StatCounter GlobalStats[1] as can be seen on Figure 2.1.

This change in the type of devices people use to access services is being reflected on the E-Government field, leading to coining the term Mobile Government, which is generically E-government adapted to mobile devices.

In their article, Kushchu et al. conclude that "M-government is inevitable" following the technological advances in the fields of mobile Internet access and World Wide Web, the value added benefits for business models in these fields and, again, the citizen's rising expectations for better and more convenient governmental services. [21]

However, the authors also state that M-government will not replace the traditional e-government approaches, but instead will be complementary to these.

[1] `http://gs.statcounter.com/`

2.2 Life Event Services (LES)

The European project OneStopGov[2] specified the life-event ontology as a way of properly representing the Life Event Service (LES) concept proposed by [22]. The project defined life-events; the active, citizen-centric approach; and the definition and use of generic workflows and reference models.

In the OneStopGov project's approach, a Life Event Service (LES) is a specialized service targeted at a given situation of a specific citizen, such as the birth of a child or buying a house. The provisioning of this service may involve the integration of several Public Administration (PA) Services, following the integrated services model described earlier in this document

The Life Events approach is particularly interesting as it doesn't rely on the person knowing exactly which public services one needs but instead focus in a particular life situation one wants to achieve - to build a house, to start a business, to get married, etc.

These services are typically made available through a life-event web portal, a centralized location where the events are orchestrated to fulfill the needs of the citizen.

2.2.1 Life Event Portals

There are two types of life-event portals: passive life-event portals and active life-event portals. Both are websites that use a topics/sub-topics structure to allow the user to search the particular life event that applies to his/hers current situation. What distinguishes them is the way they act after the user selects the event. [23]

Passive Life-event portals assist the user by providing information about what public services are required for a specific life event and what documentation the citizen needs for that service. The information these portals provide is not tailored to that citizen's specific needs. For example "the same life-event (e.g., starting a business) can differ in some aspects from user to user (e.g., starting an import/export business differs from opening a boutique with clothes). The problem is therefore that an individual life-event offers services regardless of the actual user's problem.". [23]

Active Life-event portals employ a knowledge-based system to form an active dialog with the user. [23] In these portals, user involvement is required to identify and solve problems related to particular life situations. This approach offers public services that better meet the citizen's expectations.

Despite bringing real benefits for the citizen by reducing service complexity, active life-event portals may raise the same privacy concerns described in section 2.1.2. Firstly, because Life-event Service portals are typically owned by a single entity (PA department) — which integrates several partial services to offer a single centralized service to the citizen, typically with direct communication between PA departments. [24] [15] [16] Secondly, because the personal information required for each individual service is aggregated by the centralized service provided by the portal prior to being retransmitted to the individual services that require it. This makes the centralized service have access to all the details of the citizen situation and all the information that is being exchanged with the other institutions, providing the opportunity for the services portal owner to create a detailed profile on the citizens that are requesting the service.

[2] `http://cordis.europa.eu/project/rcn/93614_en.html`

2.2.2 CHAPAS Model

CHAPAS (Citizen-side HAndling of Public Administration e-Services) is a proposal for a Life Event Service interaction model which follows a decentralized architecture where the citizen replaces the role of life-event portals in the handling of the data flows between Public Administration departments [9].

This is achieved by using an application, Chappie (Citizen APPlication to Interact with E-services), which role is to interpret the services and the interactions involved and guide the citizens throughout the process of obtaining all the documents required to solve a determinate life event.

For a service to be available in the CHAPAS Model, they must comply with a specification called Required Documents Policy (RDP). The RDP contains fields that identify the service, the documents that the service needs and how they can be obtained. More specifically, the Required Documents Policy contains the following items in the XML format:

- Identification of the service and the Institution that provides it;
- Information the user has to input;
- Requirements for the documents the citizen must provide;
- Characteristics of the documents that are produced by the service;
- Attributes used to aggregate documents;
- Circumstances relevant for the determination of what documents the citizen must provide to obtain the service;
- Instructions on how the Chappie should proceed depending on the user's answer to a question;
- Digital Signature of the service.

Chappie can obtain documents from other services or from the citizen's device, but it is the service that, in its own RDP, defines what documents are required and how they must be obtained.

Obtaining a service in the CHAPAS Model requires three steps: obtaining the service's RDP, enumerating the documents required by that service, and obtaining the service by providing it with the required documents.

When a service requests a document that is issued in another service, that other service is called a partial service. Partial services are, by definition, services, and, as such, they also specify their own required documents and where they can be obtained. Figure 2.2 shows the sequence of interactions in Chappie for obtaining the services' RDPs and the corresponding services.

As shown in that figure, it is not the service that obtains the required documents from the other institutions but instead it is Chappie that is responsible for guiding the user in doing so.

By working this way, CHAPAS ensures the citizen is always in control of what personal information is being requested at each moment and by which entity and ensures the citizen's privacy by creating obstacles on citizen profiling.

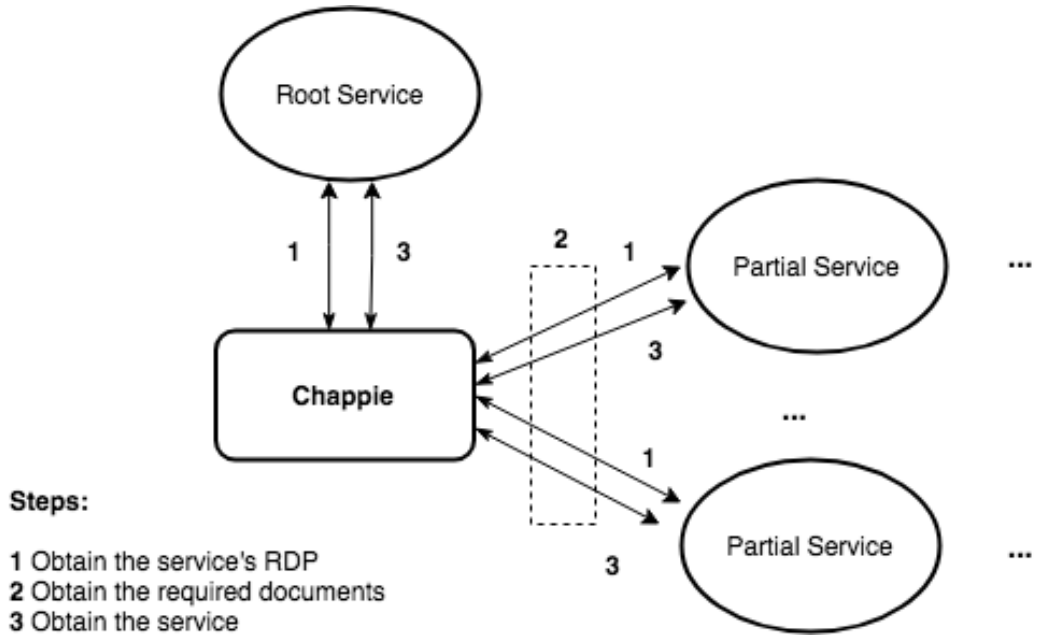

Figure 2.2: Sequence of interactions for obtaining services in Chappie (source [9])

As a way to preserve privacy, Chappie is by design agnostic on the type of documents and data that it handles, making it very limited on the extent to which tasks can be automated. This means that while some automation may be possible, it will not prevent the user from having to verify if the information was auto-filled correctly or to input the same information more than once, as Chappie does not know the content of the documents.

That agnosticism is at the same time a feature and a limitation. A feature because it ensures that this model is suitable for almost any service, regardless of what service it is and who's providing it. And a limitation because, while making CHAPAS a model that can be suitable for almost all scenarios, it may make the citizen have to input the same information more than once, as providing auto-complete features or other usability enhancements requires further knowledge on the document's content types, which can't be ensured.

2.3 User Interaction Models

Asking the user to give information to a system in a multi-step process can be achieved in different ways. Wizards, explored in 2.3.1, are the most used solution, but that are other approaches for solving this problem that are becoming significantly more used in software development, like conversational interfaces, explored in section 2.3.2 used in chat bots and virtual assistants.

2.3.1 Wizards

Software wizards, commonly known as wizards, are, according to Microsoft's Windows Design Guidelines on Wizards, applications used to guide the user in performing a single task that involve several multi-step tasks. [25] They are not a form of user interface, but they are

very useful in providing a good User Experience in several configuration scenarios such as setting up the environment where a given software will work.

The Wizard's tasks are separated in pages that contain "Next" and "Previous" buttons. A page, among others and depending on the wizard's purpose, can present information to aid the user to make a choice, present legal information like the application's Terms Of Service, or customization options on how the configuration should be handled.

Besides informational and decision-based pages, wizards also have a commit page, i.e., a page where the user is presented with the general overview of the provided information before the wizard's main task is started, since besides that point that action may not be able to be undone. During the service's main task, it is common for an indication of progress to be displayed.

Wizards should be avoided in situations where a task is not decomposed in sub-taks that must be presented sequentially. Presenting a Wizard in situations like those may lead to the user clicking "Next" several times without reading, leading to a poor overall user experience.

Microsoft states that "wizard designers often mistake users' rapid clicking of the Next button as evidence of the usability, simplicity, and integrity of their pages" and that "the ultimate wizard experience isn't Next, Next, Next, Next, Finish. While such an experience suggests that the defaults were well chosen, it also suggests that the wizard wasn't really necessary because all the choices are optional".

Microsoft also states that before creating a wizard, it should be considered whether users really must be interrupted from the main flow of the program as "there may be a lighter, inline, contextual solution that will ultimately feel more helpful and efficient to users".

Navigation guides

In some cases, a task can have so many steps that users may get lost in the sequence, or at least should be aware of how much longer the task will take to complete. For those scenarios, navigation guides can be implemented in wizards.

Navigation guides often appear as a list of pages or sections of the wizard, looking a bit like a table of contents, in a column or pane typically on the left side of each page indicating the current step, the steps the user has already completed and the remaining steps.

Navigation guides can be sequential or non-sequential. In a sequential navigation guide, the user has a predefined path he has to follow to complete the wizard, meaning he can not go forward while all the previous steps are not completed, unless they are optional. In a non-sequential navigation guide, a user is able to select the step he wants to go and can return at any time. In such scenarios, the wizard must explicitly mark what steps were already visited so the user does not get confused.

2.3.2 Conversational interfaces

Conversational interfaces are interfaces where the user is presented with the steps required for a given task as a conversation instead of other interaction method like a wizard.

There are several commercially and non-commercially available conversational interfaces on the market, like Virtual Assistants and Chat Bots.

Chatbots

A Chatbot is a computer program which conducts a conversation using voice or text input methods. Such programs are often designed to simulate how a human would behave as a conversational partner.

Chatbots can be used in several scenarios, such as providing guidance to costumers in situations they need help, display the list of services offered by a company, make on-line purchases or provide other types of information and alerts.

Some years ago, chatbots were regularly found in companies' websites. The Portuguese airline TAP used to offer a chatbot named Sofia that could answer questions related with check-in, baggage rules, services for passengers with special needs, the company's loyalty plans, amongst others. The Swedish company IKEA also used to have a chatbot, named Anna, that could answer questions related to the store products or delivery conditions. And even the telecommunications company Vodafone used to offer a chatbot named RED to which customers could make questions about the plans the company offered.

Nowadays, chatbots are more commonly found integrated in other applications, such as Facebook Messenger or WhatsApp. Newspapers like The Wall Street Journal and CNN have bots in the Facebook Messenger platform that can send news alerts to users of that application. Several airline companies also have Messenger chatbot that allows their costumers to book or check flights and boarding passes. Other companies use the technology to promote discounts or as a way to process payments to their stores.

Virtual Personal Assistants

A Virtual Personal Assistant (VPA), also known as Virtual Assistant is a software agent that can perform tasks or services for an individual by simulating a conversational dialogue between the user and the virtual agent. Even though they are similar to chatbots in the way they work, their purpose is different. Whereas the first are usually built around company-oriented services, the latter are more focused on the daily needs of the people who use them.

Examples of Virtual Assistants are Apple's Siri, Microsoft's Cortana, Amazon's Alexa, among others. These assistants can be found in mobile devices, computers and other dedicated devices such as Amazon Echo or Google Home. Interaction typically involves using the user's voice to speak to the bot, but in text-based interactions may be also supported.

A virtual assistant can perform a wide variety of tasks, including searching the web, replacing customer support, making on-line purchases, deliver news and weather information or perform more complex tasks like searching for a nearby cinema where the highest rated movie at the moment is in exhibition.

In order to fulfill their tasks, virtual assistants gather data from different on-line sources and typically run text or speech recognition engines outside of the user's device, raising some privacy concerns regarding their mode of operation.

Despite those concerns, Virtual Assistants' usage data shows their usage is growing and some analysts already picture them as replacing mobile applications in the future. Gartner expects that, by 2019, "VPAs will have changed the way users interact with devices and become universally accepted as part of everyday life". [26]

2.4 Reference Implementations

In this section we will be focusing on applications that are in some way similar or close to what is trying to be achieved with this work. User Interfaces for already existing Life-Event portals or service discoveries are of several importance, as basing a User Interface in an already known model is a good starting point for User Interface development.

We analyze one active Life-Event Service Portal, Oracle's Peoplesoft software solution, and two passive Life-Event Service Portals, Portal do Cidadão and the Paredes Municipality E-Services portal.

2.4.1 Oracle Peoplesoft eBenefits

Oracle PeopleSoft[3] is a suite of applications designed to address complex business requirements. The suite provides applications that target scenarios such as human resources management, financial management, supplier relationship management, supply services management and enterprise services automation, but for the purposes of this work, we are only interested in analyzing the eBenefits feature of this suite of applications.

The eBenefits feature of PeopleSoft allows an employee to manage his relation with the employer company regarding Health Benefits, Insurances, and Savings Plan in a Life-Event Services portal.

The application provides a portal with an interface where employees can manage their benefits situation. Enrolling in a benefit or reporting a situation where there is need for a change in the Benefits program is achieved by using a Life-Event based approach (Figure 2.3).

In a Life Event page employees can enter information about a recent life event such as marriage, child birth, adoption, or divorce, and upload documents required for the type of life event being processed.

After selecting the event, the user is greeted with an event welcome screen for a Wizard that will guide him through the tasks (steps) required for applying for that service provisioning. Tasks can have different states, depending on if they were started or not, are in progress, viewed, completed or require interaction from the user to perform an additional task (figure 2.4. Tasks can be switched by clicking the "Next" and "Previous" buttons available on the User Interface.

Whenever the user finishes a task, he is prompted with an informative screen saying the information he introduced was saved. This is important, as it gives the user insurance that whenever he resumes that service later on, he will be able to resume where he last left off instead of having to start the service all over again.

The application also lists on the tasks pane information on what user information is being requested by the overall event, as can be seen on Figure 2.5.

Before finishing, the user is able to review all the information he submitted and can also find related content to that service, including information provided by the company regarding the list of benefits that are available or links to external pages where the user can find more information.

On event completion, additional related information is given to the user. In the scenario explored in the source video of Figure 2.6, the user is informed about related tasks that may be required that are outside of the scope of the service. This is useful to the user, as it prevents

[3] `https://www.oracle.com/applications/peoplesoft/index.html`

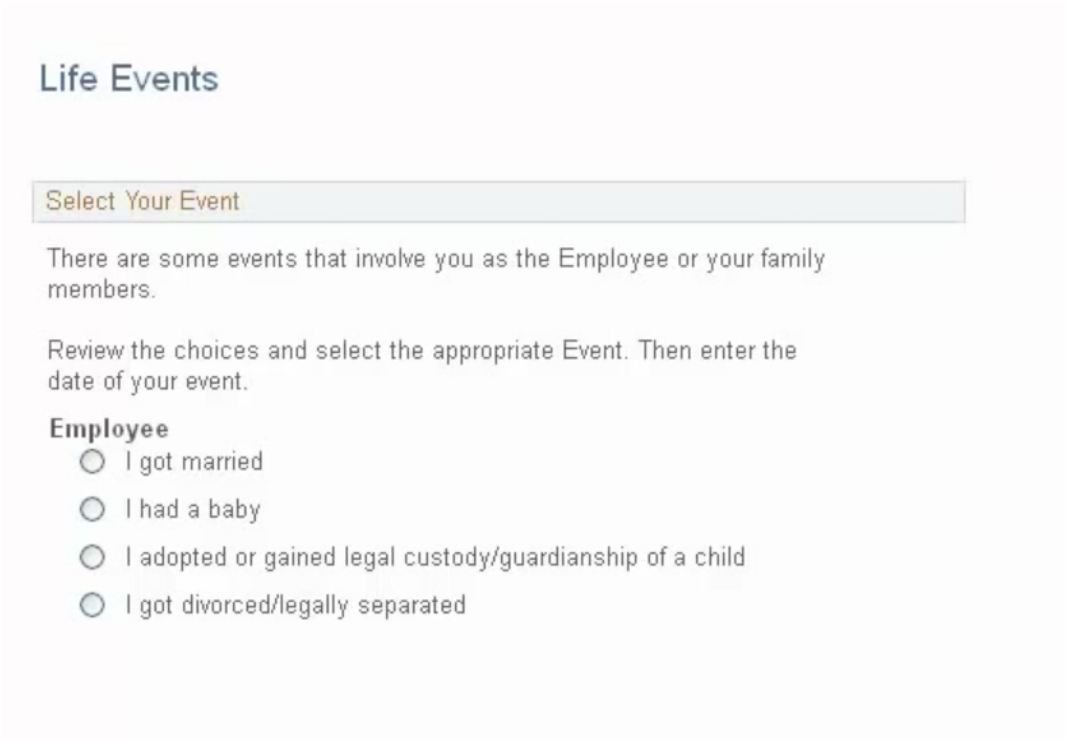

Figure 2.3: Oracle PeopleSoft: Select Life Event Screen (source: [27])

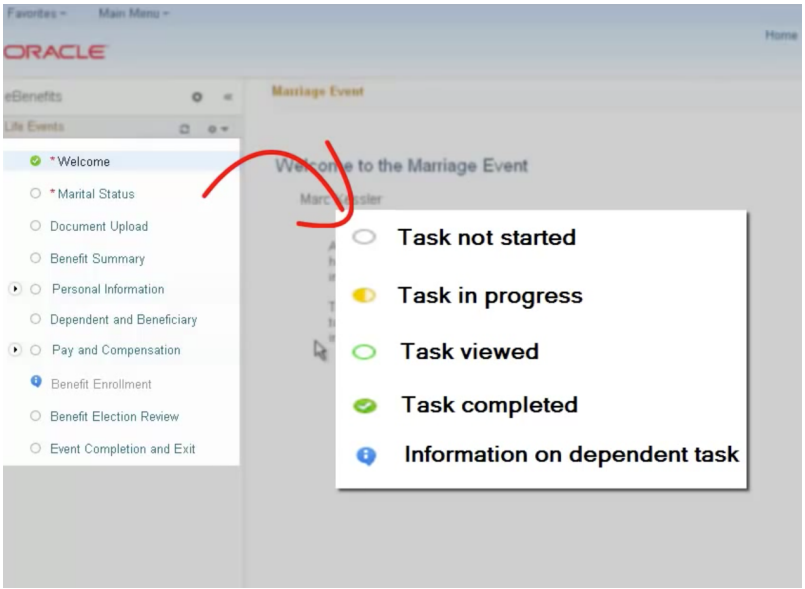

Figure 2.4: Oracle PeopleSoft: Task Status (source: [27])

Figure 2.5: Oracle PeopleSoft: Personal Information Sub-task (source: [27])

the user from having to deal with potential legal or personal consequences that may derivate from the service usage.

Peoplesoft has a set of interesting features that can be used in this work, namely the color and symbol information on tasks status, the visual information on the amount of steps required to finish the task and the related information, both on service review and on service completion.

In the next section, another Life Event Services Portal will be analyzed, Portal do Cidadão.

2.4.2 Portal do Cidadão (Citizen Portal)

The Portuguese Citizen Portal, Portal do Cidadão[4], is a Life Event Services portal that offers the list of the Public Administration services that are electronically available to citizens and entrepreneurs.

The services are accessible both using a search bar or by selecting a category of life events. These categories are associated with the daily life of people and organized in order to optimize the discovery of services when searching for a topic and not by any specific service.

For each service available in the portal, there is a service description and information about the responsible entity, who the service is targeted at, where it can be obtained, the conditions required for the service to be obtained (including required documentation), total cost, service time, among others, as can be seen on figure 2.7.

Besides listing services and entities, the portal also features a list of guides on how to obtain certain documents or proceed in a given daily situation, making it a primary point for searching all Public Information.

There is also a section for service providers, where information about each department or institution is presented. That information includes the institution mission, ministry to which it belongs, organic law and the available contacts.

[4] `https://www.portaldocidadao.pt/`

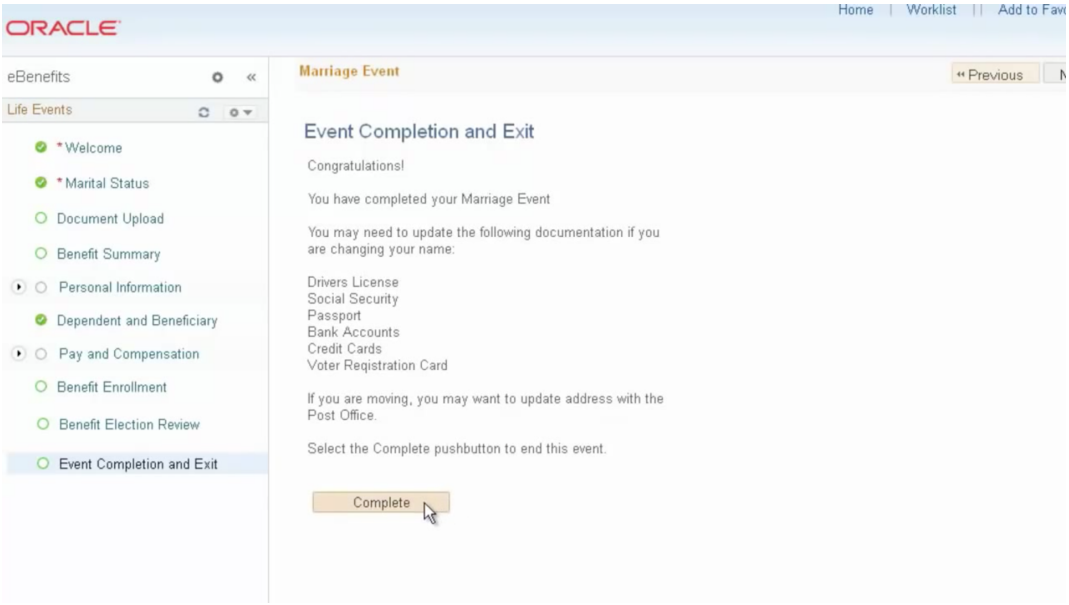

Figure 2.6: Oracle PeopleSoft: Complete Event and Exit Screen (source: [27])

Figure 2.7: Excerpt from a service page in Portal do Cidadão

Figure 2.8: Documents bag in Portal do Cidadão

The Citizen Portal also features a wallet section, where the citizen can store personal documents that can later be reused during service provisioning. Documents can be uploaded from a web browser and can have several attributes with personal identification and information where they should be stored, as can be seen in figure 2.8. The wallet also features the ability of importing data from the Citizen Card to create an identification profile.

Services are not provided through this portal. It serves only as a link for other websites (specifically, the website of the provisioning institution) where that service can be provisioned.

Even though the portal not offering services directly, the services to where it redirects the citizens follow a whole-of-government model. Most of those services are provided by a single entity that can gather information about the citizen that is using them from other entities.

There are some key concepts presented in this portal that are very useful for our work, such as the layout used to present the available services, the wallet functionality, and the calendar.

Next, we will explore a more simple Life Event Service portal, the Paredes Municipality e-Services portal.

2.4.3 Paredes Municipality e-Services portal

The Paredes Municipality provides an on-line services section in their website where residents can apply to different municipal services.

Even though the portal itself is a Life-Event portal due to services being organized accordingly to Life-Event categories (Figure 2.9), the services provided are not life-event services because the citizen does not run a single service that integrates other services to fulfill a situation in his life but instead is only presented with the list of documents he must obtain prior to apply for the service, as ilustrated in Figure 2.10.

This portal is also considered a passive Life-Event portal because the information that is presented to the user, more specifically, the list of documents that are required to fulfill a given situation is not personalized according to the needs of the citizen, having this to know if that given document is required or not.

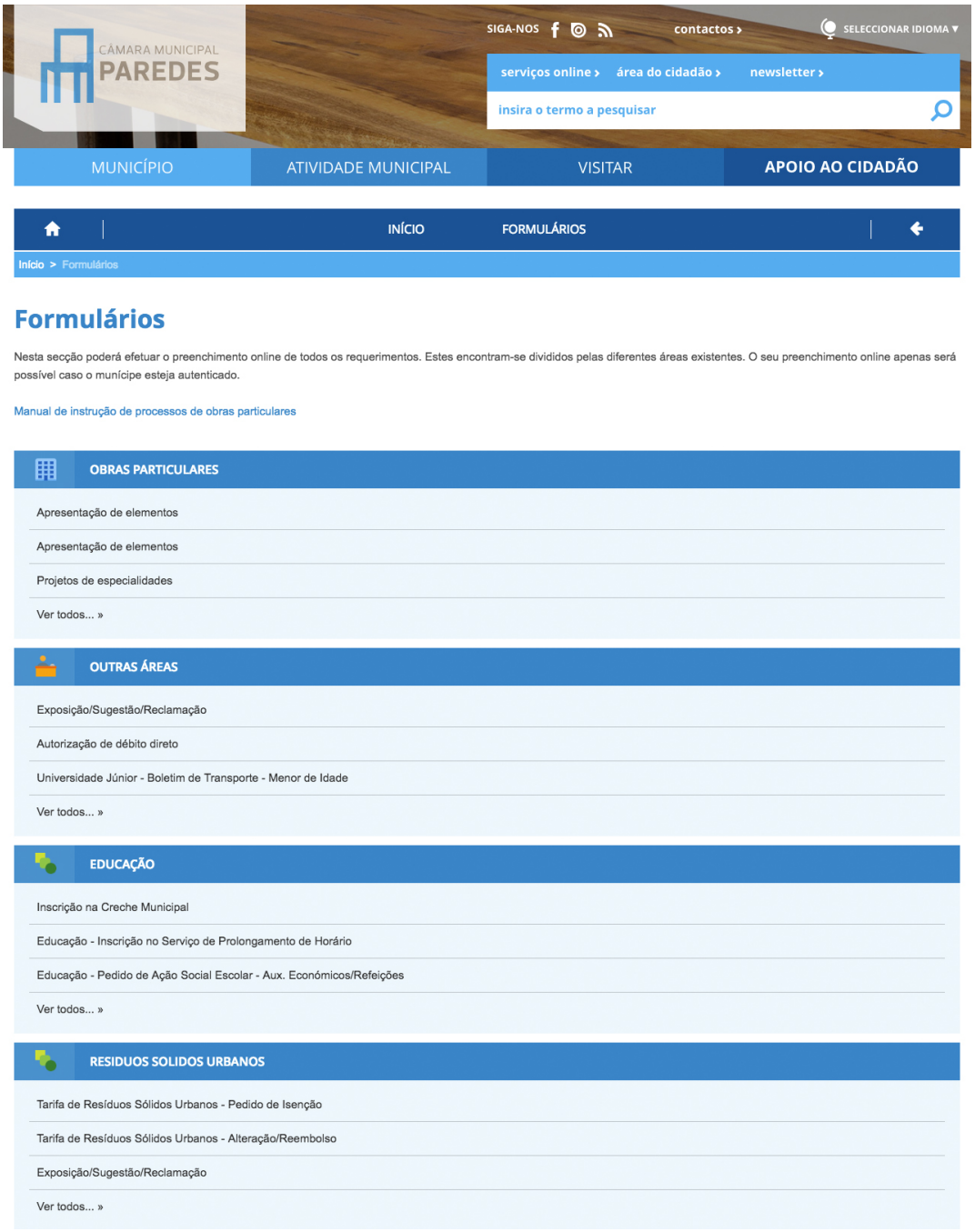

Figure 2.9: Excerpt from the Forms Page in Paredes Municipality's E-Services Portal

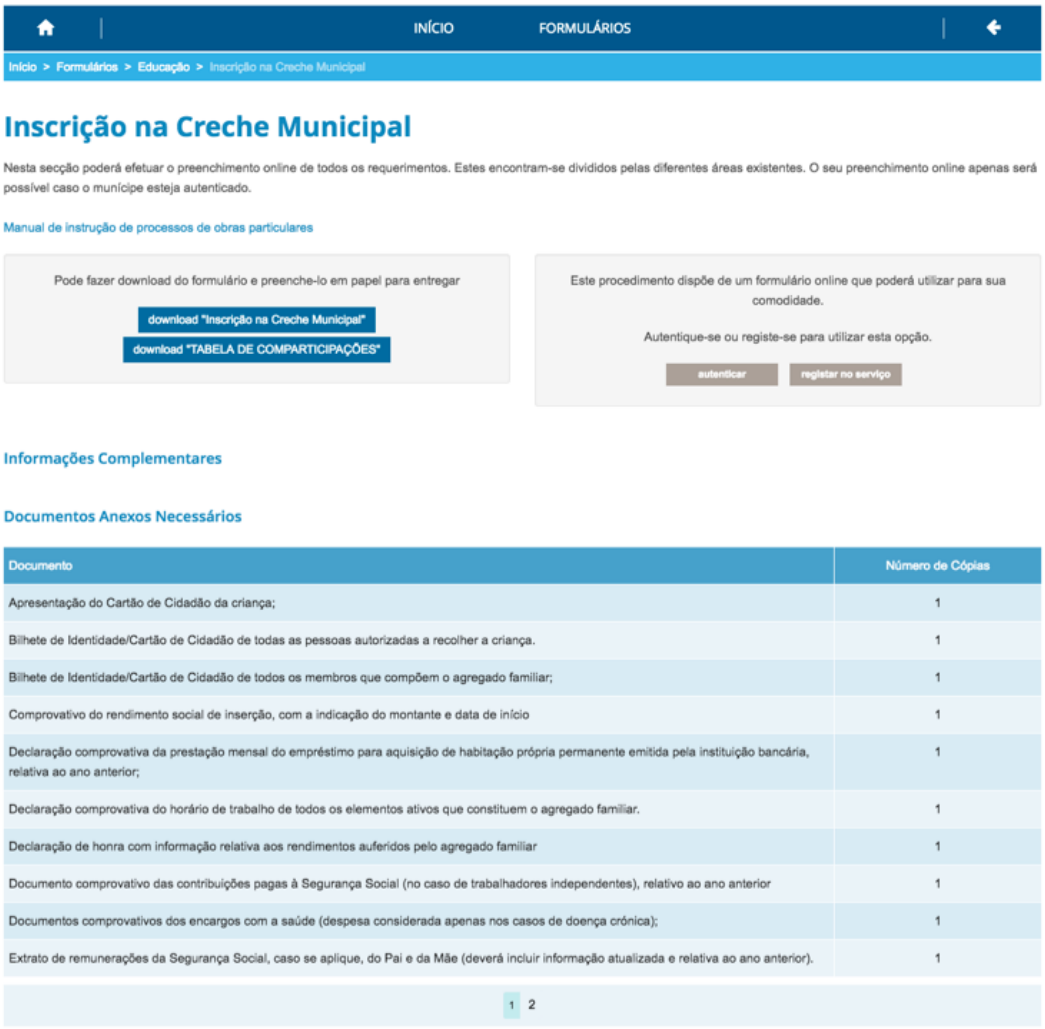

Figure 2.10: Excerpt from a service page in Paredes Municipality's E-Services Portal

Portals like this one are useful because they provide a centralized place where the citizens can view the documents they must provide in a given situation. However, the lack of a proper service provisioning model with step-by-step instructions and services tailored at the specific needs of the citizens make them a not very useful example to this work. Some features like the ability to download the required forms to apply for a service or the description of the context where the service provisioning is applicable are interesting. However, those features are outside of the scope of this work.

2.5 Usability Evaluation

This section will focus on important aspects that should be taken into consideration when designing a User Interface. They serve as a generic starting point for every web user interface and not exactly for the specif type of interface we want to develop in this work.

Following user interface best practices is the ideal way to captivate the attention of the user and to avoid developing an interface that can leave the user frustrated or the user anxiety while using the application.

2.5.1 Usability Standards

It is often assumed that a standard has the same meaning as a precise specification. However, setting a standard in Human Computer Interaction (HCI) is not always easy. Despite standard user interfaces providing the benefit of consistency, they become out of date as technology changes and are usually only appropriate for limited types of users and tasks. [28]

Although many design guides providing a detailed specification of the nature of the user interface, most work on international standards for HCI has not been about precise specification, but instead has concentrated on the principles which need to be applied in order to produce an interface which meets user and task needs. [28]

The U.S. Department of Health & Human Services specify a list of Usability Guidelines () which "are research based and are intended to provide best practices over a broad range of web design and digital communications issues". [29]

Guidelines are classified in scales from 1 to 5 according to "Relative importance" and "Strength of Evidence". Values for these ratings were based on the evaluation of 16 reviewers (half Web site designers and half were usability specialists) and 8 usability researchers, practitioners and authors, all published researchers with doctoral degrees, experienced peer reviewers, and knowledgeable of experimental design, respectively.

According to these guidelines, the most Important Usability Guidelines According To Strength of Evidence are the following:

- Provide Useful Content
- Standardize Task Sequences
- Design for Working Memory Limitations
- Align Items on a Page
- Use Descriptive Headings Liberally
- Use Black Text on Plain, High-Contrast Backgrounds
- Use Attention-Attracting Features when Appropriate
- Use Familiar Fonts
- Emphasize Importance
- Order elements to maximize User Performance
- Use Data Entry Fields to Speed Performance

- Use Simple Background Images
- Use Video, Animation, and Audio Meaningfully
- Use Images to Facilitate Learning
- Use Mixed Case with Prose
- Group Related Elements
- Use Color for Grouping
- Use an Iterative Design Approach

Aside from these guidelines, there is ISO 9241, a multi-part standard from the International Organization for Standardization (ISO) that covers ergonomics of human-computer interaction.

Part 151 of that set of standards (ISO 9241-151), "Guidance on World Wide Web user interfaces" [5] provides recommendations for the user-centered design of web user interfaces. The recommendations cover much the same scope as HHS Guidelines, but are documented in a more concise format appropriate for an international standard. [30]

However, as some topics of the development process and evaluation are already covered by other ISO standards — such as ISO 13407, ISO TR 16982, and ISO 9241-11 for Design Process and Evaluation, ISO TS 16071 and WAI Guidelines for Accessibility, ISO 9241-12 for lists, ISO 14915-2 for widgets, and ISO 14915-3 for Graphics, Images, and Multimedia —, the purchase of several standards is required for a complete guidance on the web that follows the ISO Standards specification. [30]

Moreover, apart from the costs associated with the standards purchase, some interpretation is needed to understand what specifics on each standard are applicable to a given situation. Because of these limitations and because the HHS usability guidelines are publicly available for every person to read, those guidelines were the ones taken as the reference to develop the application proposed in this work.

2.5.2 Heuristic Evaluation

An heuristic evaluation is a usability inspection method used in software development that helps to identify usability problems in the user interface design.

Jakob Nielsen defined 10 general principles for interaction design called usability heuristics. [31] These principles are not specific usability guidelines but instead broad rules of thumb that should be taken into consideration when designing an application.

According to the Nielsen's Heuristics, applications should evaluated taking into consideration the following factors:

- Visibility of system status
- Match between system and the real world
- User control and freedom

[5] `https://www.iso.org/standard/37031.html`

- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help users recognize, diagnose, and recover from errors
- Help and documentation

Visibility of system status means the system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Match between system and the real world means the system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. The system should follow real-world conventions, making information appear in a natural and logical order.

User control and freedom means that if a user choose a system function by mistake, the system should provide a clearly marked "emergency exit" the user can use to leave the unwanted state without having to go through an extended dialogue. This heuristic also includes providing support for undo and redo operations.

Consistency and standards means users should not have to wonder whether different words, situations, or actions mean the same thing. The application should define a set of conventions and follow them in all different scenarios.

Error prevention means that the application design should focus more on avoiding problems from occurring instead of providing good error messages. The application should either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

By consistency and standards, the author specifies that applications should try to minimize the user's memory load by making appropriate objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Also, instructions to use the system should be visible or easily retrievable whenever appropriate.

Flexibility and efficiency of use means that in some situations the use of accelerators may speed up the interaction for an expert user while at the same time not confusing novice users, who sometimes do not see them. They should allow users to perform frequent actions with more ease.

By aesthetic and minimalist design, Nielsen means that dialogs should not contain information which is irrelevant or rarely needed. He goes as far as saying that every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Help users recognize, diagnose, and recover from errors means that error messages, when presented, should be expressed in plain language (with no codes), precisely indicate the problem, and constructively suggest a solution.

By Help and documentation, Nielsen means that even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

2.5.3 Cognitive Walkthrough

The way developers expect users to performing a given task may not be the most intuitive way of doing so. Buttons that allow the user to do the action they are expecting may be visible but not understandable or the application may not display enough visibility on its status that allows the user to understand an action was successful.

Cognitive walkthrough is a method for identify usability issues by analyzing how easy it is for new users to accomplish tasks with the system. [32] The method consists on creating a list of tasks the users can perform in the application and, for each task, describe a sequence of sub-tasks users have to do to accomplish it, and the expected system response to those actions.

According to the method, after the tasks being enumerated, people familiar to the project should go through each of the task using the application and ask themselves the following questions:

- Will the user try to achieve the right effect?
- Will the user notice that the correct action is available?
- Will the user associate the correct action with the effect that the user is trying to achieve?
- If the correct action is performed, will the user user see that progress is being made toward solution of the task

When developing a User Interface, conducting a cognitive walkthrough evaluation is important because it helps identify usability problems that other methods such as heuristics evaluation may have missed.

2.5.4 Usability Tests

Usability tests are tests done during the development of the application that, unlike the previously mentioned methods, involve users in the evaluation of the application to verify if the proposed features and ideas are well understood and accepted by potential users of the application.

Performing an usability test requires organizing a group of users (evaluators) willing to test the application and and assign them a list of tasks they have to perform.

While using the application, evaluators are monitored by people who take notes on their behavior when performing each task in order to monitor the evaluator's behavior and understand what can be improved in the application.

Chapter 3

Chappie

This chapter presents a technical overview of the state of the Chappie prototype when this work was started. Section 3.1 covers how the application starts a service and how that service's RDP is processed. Section 3.2 describes the process of obtaining a service.

This chapter also enumerates usability problems this prototype had (section 3.3) and the technical challenges in adapting this prototype to a more modern application (section 3.4).

3.1 Building the Service Dependencies Tree

It was already said earlier that in the CHAPAS Model services are described according to a specification called Required Documents Policy, which lists the documents that a service needs and how they can be obtained. It was also said that required documents are always handled by the citizen but can be obtained from several sources defined in this specification.

When given an URL for a service, Chappie reads the service's RDP from that URL and tries to identify all the required documents for the service. This list of required documents is called a dependencies list. If a required document has to be obtained from another service, in a called a Partial Service, Chappie recursively repeats the process described before until all the required documents are identified (Figure 3.1 shows a simple example for an hypothetical service named 1A. In this example, combinations of letters and numbers (e.g. 1A) represent documents obtained from services services. Roman Numbers represent represent documents provided from the citizen's device.).

Visually, these interactions can be represented in a tree of dependencies between services that can grow exponentially both in depth and in breadth, where the first documents to be obtained are those that are lowest in the hierarchy (Figure 3.2).

Because services in CHAPAS are Life Event Services, sometimes this process interaction from the citizen is required to continue building the dependencies tree. Situations where citizen intervention is required are those where he is free to select the provider he/she wants to obtain the document from, like the bank that can issue a proof of payment, or when more information is required about the context the service is being executed.

3.1.1 User Interaction During The Tree Building Process

When the citizen is in possession of the information about all the documents required for all the services, the dependencies tree is complete, the user may start the process to obtain

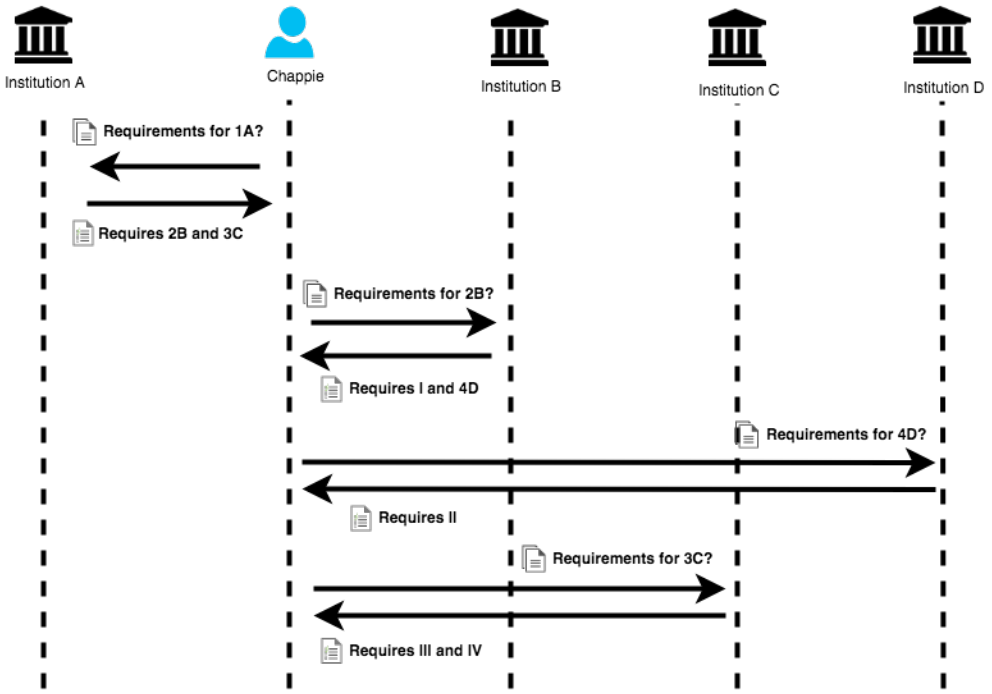

Figure 3.1: Process Chappie uses to create a dependencies tree

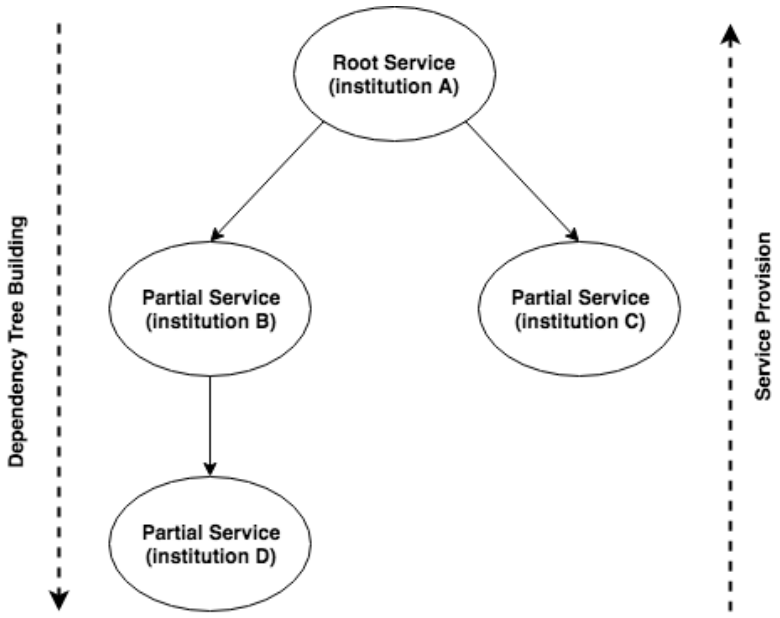

Figure 3.2: Illustrative example of a dependencies tree for an hypothetical service named 1A.

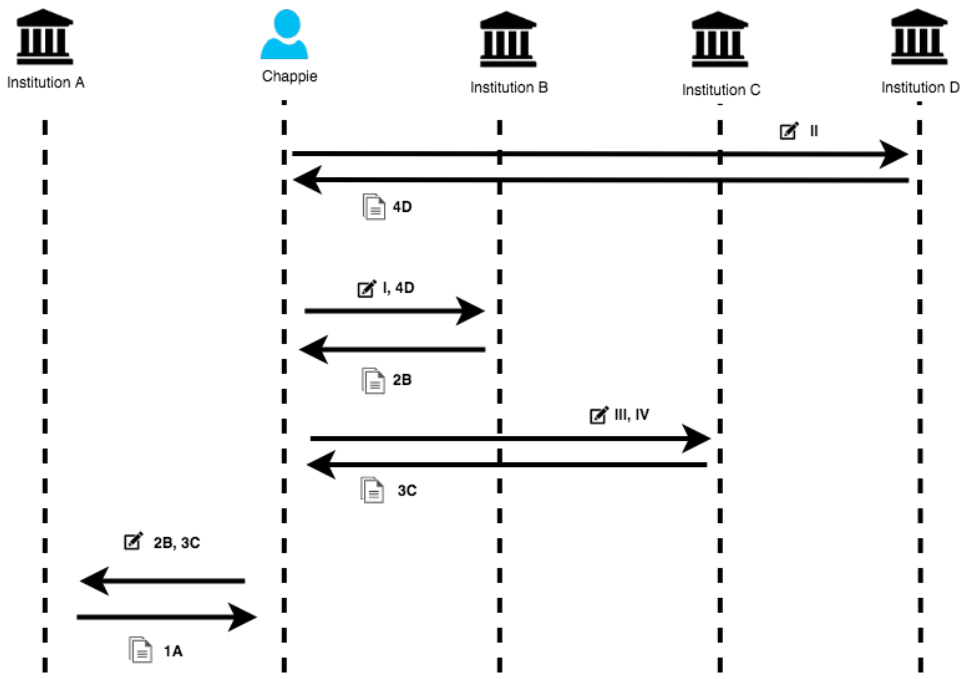

Figure 3.3: Process for obtaining services in the CHAPAS Model

---

them all till eventually obtain the root service that marks the obtainment of the complete life-event service. This process happens in the reverse order than that to build the dependencies tree (Figure 3.3 shows an example for an hypothetical service named 1A. In this example, combinations of letters and numbers (e.g. 1A) represent documents obtained from services services. Roman Numbers represent represent documents provided from the citizen's device.).

The process of obtaining a service in Chappie is mostly done automatically with little interaction from the user. However, as Chappie provides services in a similar way they are provided in active Life Event portals, user interaction may be required in some situations. For example, when more information on the context the service is being executed is required.

These situations, called Circumstances, may influence the documents a citizen has to provide to the service for provisioning. [33] As an example, when starting a marriage process, the process can be different depending on whether one of the spouses is from another country: additional documentation on that person may be required, like a passport, that would otherwise not be needed.

In CHAPAS, Circumstances are a set of questions that the citizen must answer with one of the predefined answers described by services in their RDPs. The institution that asks the circumstance is not informed on the response the user gives. Instead, the RDP must also specify a set of rules Chappie should use to evaluate the citizen response and enumerate the required documents for continuing the build of the dependencies tree.

Aside from circumstances, user interaction is also required when the RDP states that a document from another service is required but does not specify explicitly the service that the citizen should use to obtain it but instead only specify a category of services that can issue that document. For exemple, when paying for a service, the citizen should be able to select what bank he wishes to use to process the payment.

In those situations, the citizen should be able to select the issuer service and institution he wants to use to obtain that document with no restrictions in the options that are given.

3.2 Obtaining Services in Chappie

After all the requirements are enumerated, the user is then able to start obtaining all the required documents from other services and forward them to the other services that require them (Figure 3.3).

Like the dependencies enumeration process, the process of obtaining a service is partially automated but there are still situations where interaction from the user is required. These will be described in the next sub-section.

3.2.1 User interaction during the process

Not all the information from the citizens can be obtained from services. Sometimes the user has to manually insert information, like a first name. Other times, institutions may not be using the CHAPAS Model for service provision but their documents be required by other services that use it.

Chappie allows citizens to input information by typing it using the keyboard, importing or generating it using a smart-card reader, or through the upload of documents. However, neither the citizen or Chappie have control over what method to use as that is described in the service's RDP.

There are two moments when user interaction is required during service provisioning: before starting obtaining the service and as a request for additional solicitation.

Making a comparison with real life scenarios, providing information before starting provision is the equivalent of physically filling a form in an institution, attaching the required documents and picking a card for making the payment for the service. Following that same idea, the moment the payment terminal asks for the person to insert his PIN is an additional solicitation: something that is not part of the service but is necessary for it to be performed and has to be requested after everything is ready.

3.3 Usability Problems

The old Chappie prototype was a very basic application built as a tech demo for demonstrating the potentialities of the CHAPAS Model. It was not intended to be a proposal for an actual application, but it is important to analyze it as a way to understand how the different concepts were put together. However, an analysis of this prototype's usability weaknesses may prevent some potential errors in the new application.

The old Chappie prototype made a clear distinction between each stage of service provision in the CHAPAS Model. There was a clear separation between the construction of the dependencies tree and the service provision. That clear distinction when applied to the user interface, made the experience feel loose, transmitting the sensation that there was no connection between the elements that are appearing on the screen.

In the prototype, as soon as an RDP was read and if user interaction was required for answering a circumstance (Figure 3.4) or selecting a provider (Figure 3.5), a dialog box would

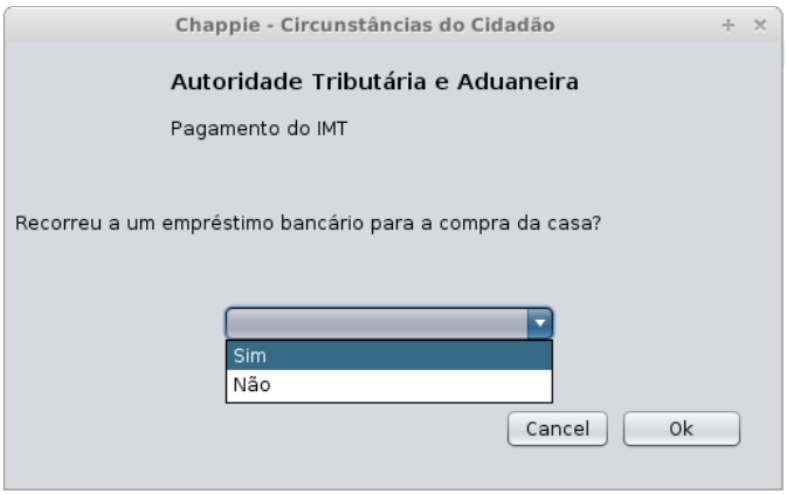

Figure 3.4: Circumstance input in the old Chappie prototype.

---

appear asking the user to make a choice even before any information about the service was visible.

Both the name of the entity that was requesting the interaction and the name of the service associated with it were present, but as there was no information on what was happening, these requests for interaction seemed to be coming from nowhere.

The same problem was felt during service provision. As said before, both stages of interaction were clearly separated: the construction of a dependencies tree was one task while the provision of the service was another. After the tree being finished, the user had the opportunity to explore the services involved in the interaction and view what documents were required.

As soon as the service started to be obtaining, new dialogs asking for interaction similar to the one on Figure 3.6 appeared to ask for input of information from the user.

After the service was finished, the tree list was updated to include all the dependencies as well as list the documents that were obtained (Figure 3.7).

It is evident that most of these problems were caused from the prototype being a tech demo and not a real application. However, the way information entry was handled was specially unpleasant even though it required the minimum amount of clicks possible in this model.

Chappie was designed with the same concepts that Active Life Event Portals are designed: ask the user a set of questions in the beginning, provide a form for inserting information or adding attachments, and present the end result. However, unlike in those portals, services in Chappie are generic and have the same behavior no matter when provided as a standalone service as a requirement for another service. This lack of a global service context poses a very big challenge in creating a User Interface because messages have to be generic and can not be tailored to the service that is currently being executed.

These problems show that Chappie can not be modeled after a traditional Life Event Portal and requires a more detailed analysis on its requirements in order to create the best possible user experience.

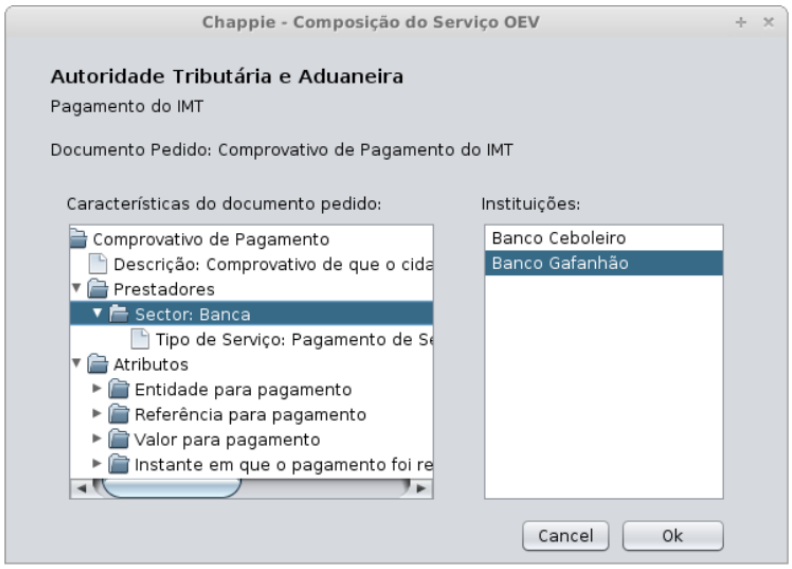

Figure 3.5: Provider selection in the old Chappie prototype.

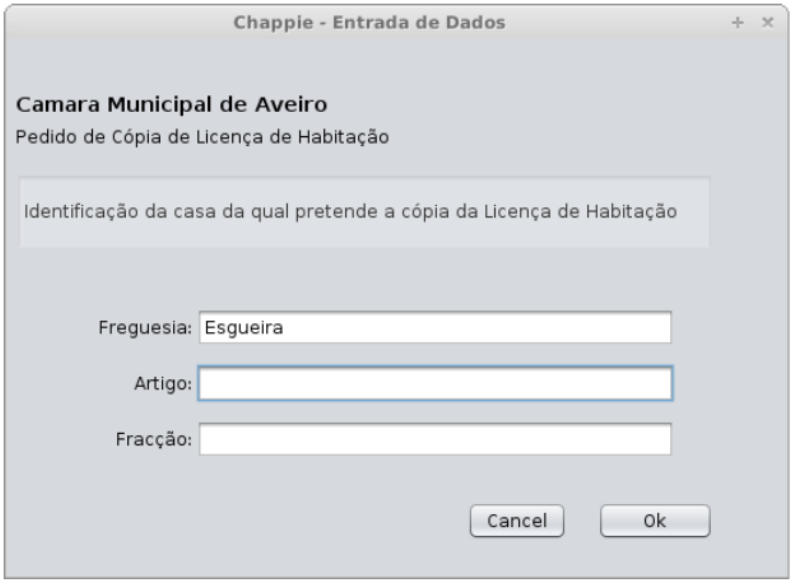

Figure 3.6: Data input screen in the old Chappie prototype.

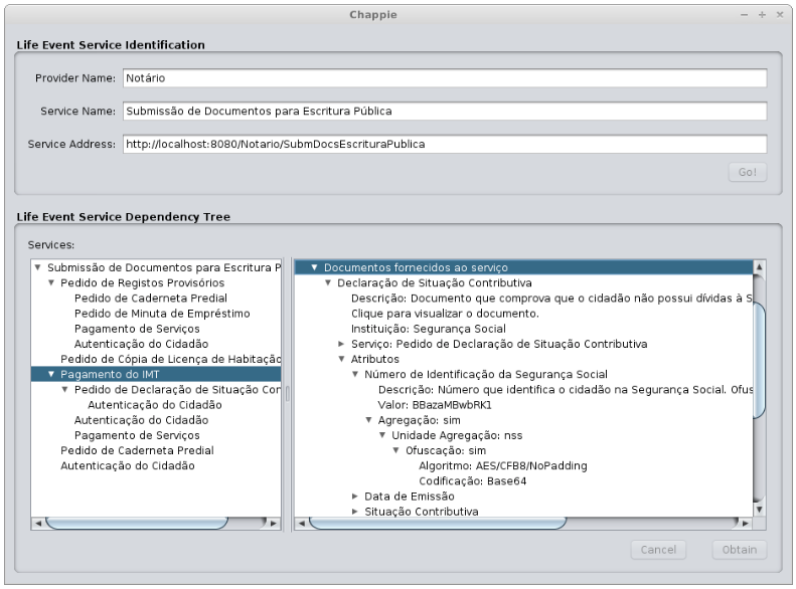

Figure 3.7: An obtained service in the old Chappie prototype.

3.4 Technical Challenges

The old Chappie prototype was an application written in Java using the Swing GUI widget toolkit for the graphical user interface (GUI).

Swing is part of Oracle's Java Foundation Classes (JFC), an API for providing a GUI for Java programs. Swing provides several UI Components for applications, such as as buttons, check boxes, labels, tabbed panels, scroll panes, trees, tables, and lists.

Swing components are written entirely in Java and therefore are *theoretically* platform-independent. An emphasis was put in theoretically because running User Interfaces written in Swing still requires the host platform (i.e., the computer, tablet, smartphone, or other device where the application is running) to have the Java Virtual Machine (JVM) - the environment where Java applications are run - installed.

At the first glance, it seems to make sense to start working on the existing prototype and build the remaining features on top of it. However, even though Java is available for several devices ranging from desktop to servers, there is no native support for it in smart-phones and tablets nowadays. None of the major Operating Systems currently available on the market support it, and the smaller ones who do are only capable of running Java ME applications, which are completely different from typical Java SE ones.

One can argue that because Android applications are technically Java applications, the current implementation code could theoretically be ported to the Android Software Development Kit (SDK) and with little changes run on this platform. However, because Chappie has several third-party requirements and has most of its codebase already written, porting it would require a significant amount of effort that would go beyond of the scope of this work. Besides, porting the application would allow it to run on Android devices but would not solve the problem of the portability amongst systems that was described earlier, meaning a more agnostic solution had to be considered.

Besides the problems mentioned above, the application had a more severe problem that needed to be addressed: it relied on synchronous execution. That means that whenever attention was required by the user — in circumstances, selecting a provider or inputting data —, the whole application flow would stop until the user had inputed all the information required for the flow to resume (Figure 3.8).

Although this approach was a standard procedure for most traditional desktop applications, it is nowadays discouraged in favor of an event-driven asynchronous approach that allows better and more dynamic User Interfaces.

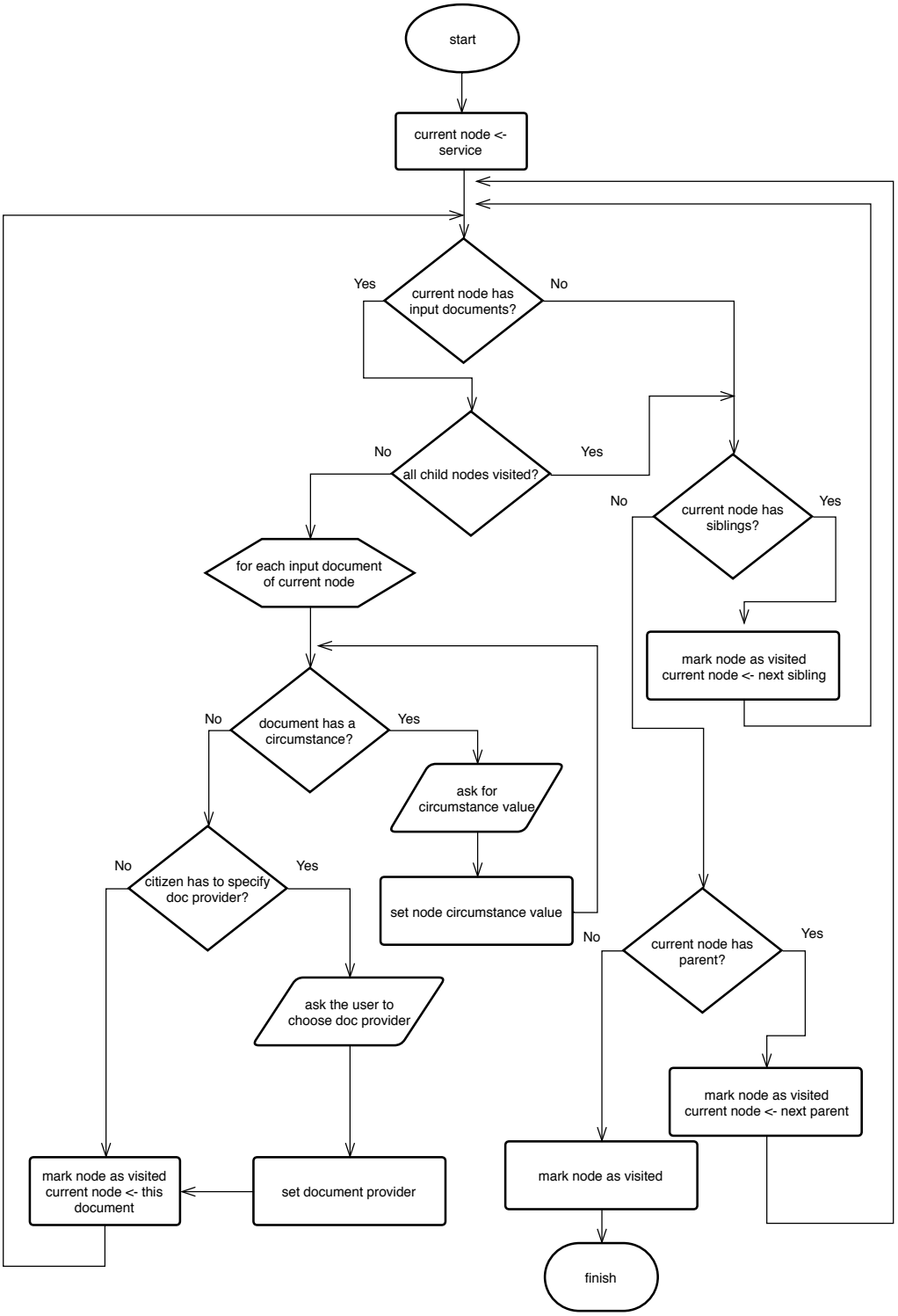

Figure 3.8: Workflow followed by the old Chappie prototype to create a dependencies tree and list all the required documents.

Chapter 4

Prototyping the User Interface

This chapter describes the aspects that were taken into consideration for developing the new user interface for Chappie and the tests that were made with users to assert their appropriateness for the application being developed.

Section 4.1 describes the profile of the typical user of this application. The features that the application should provide to its users and the usability goals are described in sections 4.2 and 4.3, respectively.

The chapter also covers the expected execution environment and its limitations in section 4.4, and the limitations that the application has in section 4.5.

Lastly, this chapter also describes the two usability tests that were conducted with users and their respective results in sections 4.6 and 4.7.

4.1 User profile

The application targets citizens who need to apply to a Public Administration service for solving a given situation in their lives. Due to the broad range of users that are covered in this profile, the only estimation that can be made regarding the expected ages of the users is them being over 18 years-old.

Regarding the expected computer literacy and previous experience the users should have, an average computer literacy is required and previous experience with e-government services may benefit the experience of interacting with the application.

Users are expected to interact with Chappie in very specific scenarios, meaning the application should rely mostly on already known and common concepts, as trying to teach the user new ones will likely increase the application's rejection rate.

4.2 Expected Features

In this section we describe a list of features that were considered important for the purpose of the application that is being developed. The purposed features include an initial screen from where the user is able to start a service, a personal wallet where he can store personal documents or documents obtained in services and an history page.

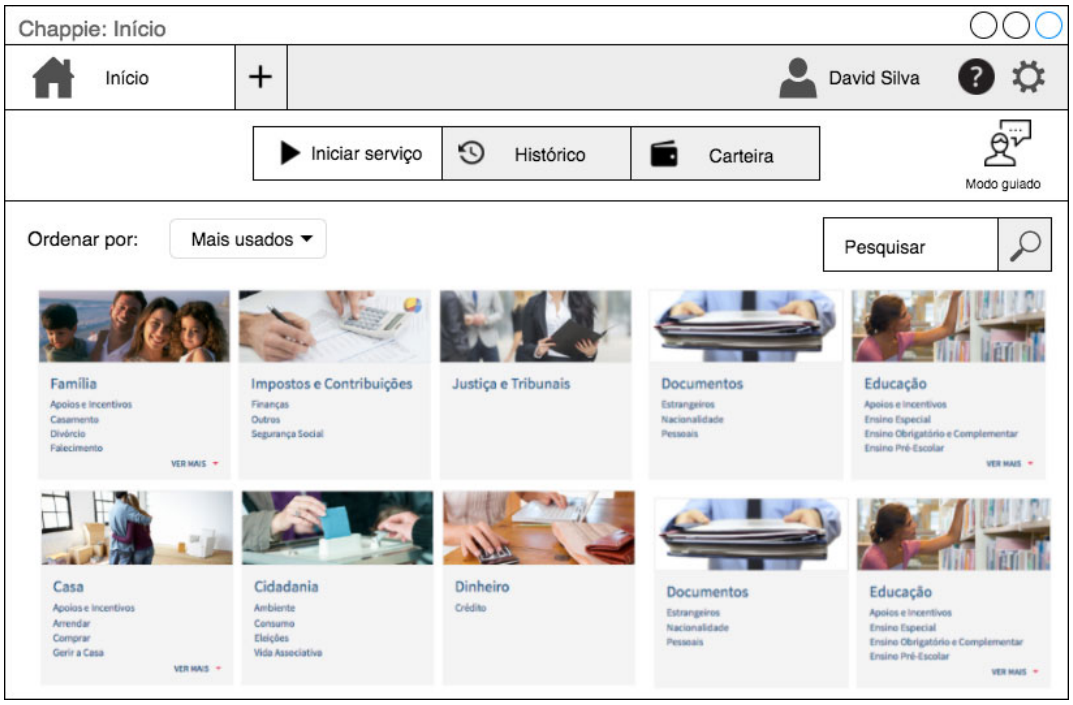

Figure 4.1: Early concept of the Chappie home screen.

4.2.1 Home Screen

The home screen is the first screen the user sees when starting the application. From that screen, the citizen should be able to easily start the root service required for solving a particular life event. Figure 4.1 displays the initial idea for this screen, which eventually suffered some changes throughout development. The dummy list of services was adapted from the "All Services" page of the Citizen Portal.

This initial screen should also provide a search bar for making it more easy for users to find the services that hey are in the need of.

While not important during prototype testing, a decision had to be made regarding how Chappie obtains the list of services available to the citizen. Several hypothesis were considered such as integrating an external portal in Chappie, or having the list of services as a local file that synchronized with an external server.

Because it makes more sense in the agnostic model of the application not being dependent on an external service, it was decided that it is the citizen who controls life events, services and groups that are available in the application. These can be manually inserted and edited, or can be imported by scanning a QR Code.

4.2.2 Citizen Wallet

The citizen wallet, hereby referred to as wallet, is a storage area where documents obtained from services or added by the citizen are stored. Documents in the wallet can be viewed by the citizen or used in services when required (Figure 4.2). Figure 4.3 shows an early concept of selecting a document from the wallet to be used in a service.

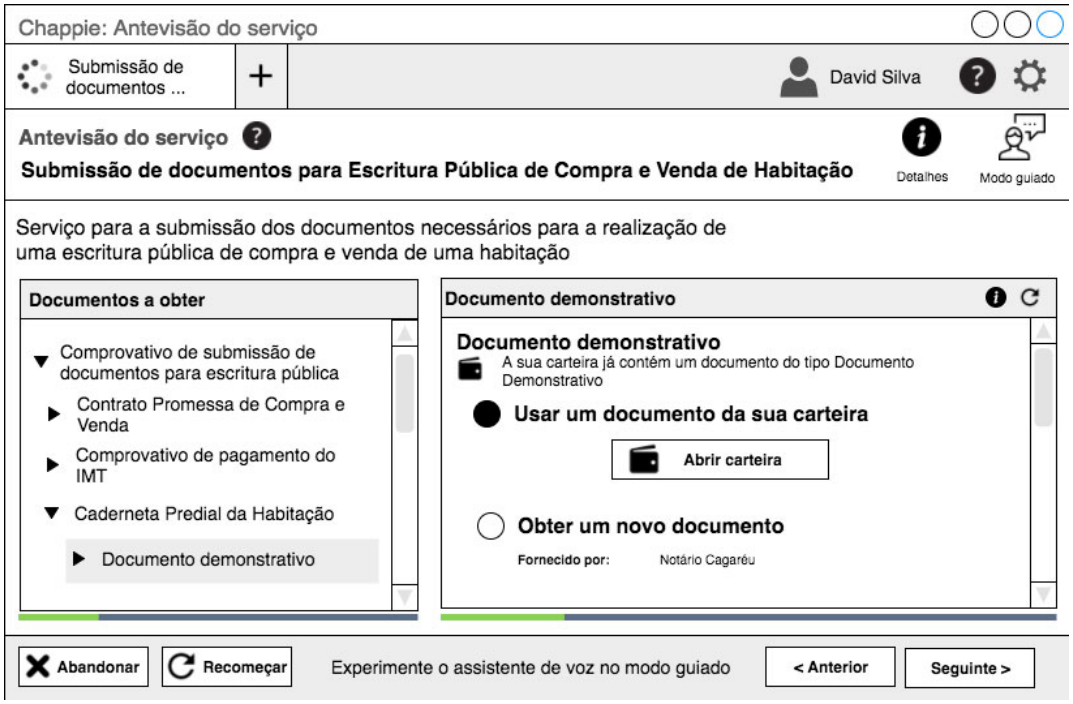

Figure 4.2: Early concept of a screen where the citizen could select to reuse a document from the wallet.

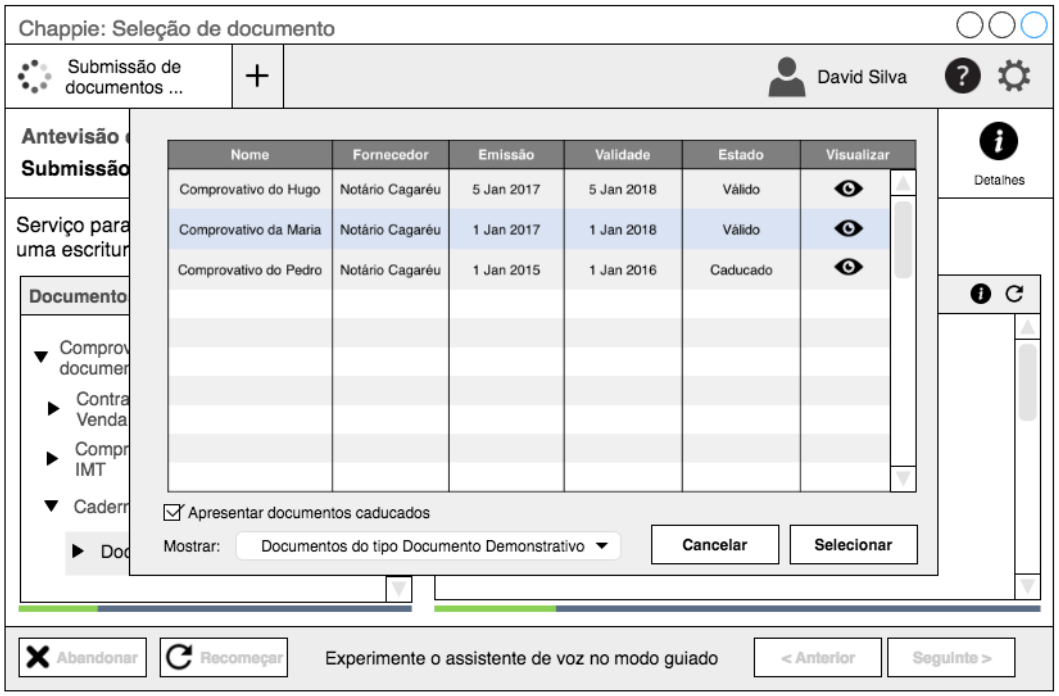

Figure 4.3: Early concept of screen for selecting documents from the wallet.

In later stages of development, the wallet started supporting more features and, in the final application, besides storing documents, the wallet also stores several identity and service cards that can be manually inserted or edited, or imported by scanning a QR Code.

Identity cards are cards that the citizen can create that hold portions of his personal information. They are a suitable way of storing groups of portions of the citizen personal information in an organized way.

Service cards are cards institutions can give to citizens that contain the specification for a service and the life events that can start it. These cards can be used both for starting Life Event Services or in situations where the citizen has to select a provider for obtaining a document.

The concept of wallet is explored in all the prototypes and was extremely well received by the evaluators and, as such, it was also included in the Chappie application.

4.2.3 History

Citizens should have a way to check what were the last services that were performed in the application. Besides listing the services in reverse provision date, the history page should display what documents were obtained during that service.

The history feature was very well received during prototype testing, in some cases being the first choice for users who were looking for the documents he had obtained previously in services.

4.2.4 Use of a wizard view to guide the users

Due to the complexity that a service in the CHAPAS Model has (the model consists in services providing documents and requiring documents that are provided by other services), it is very important to choose a navigation method that does not frustrate the users.

The usability problems of the previous prototype show that an approach with minimal interaction from the user leads to an unpleasant experience because, due to the complexity of CHAPAS requiring more information to be displayed on screen.

Thus, a decision was made to follow an interaction model similar to the one used in Oracle's Peoplesoft: a wizard. Despite the increase in the number of steps required for obtaining a service, the feedback from the users in every usability test that followed this model was extremely positive, whereas in other interaction models most users felt confused.

The wizard's steps changed several times during the course of development of the application, as can be seen in the screenshots of the different prototypes used in usability tests and in the final application.

4.2.5 Automatic information input

Chappie is agnostic regarding the content of documents or the information that is inserted by the user. It does not understand the content types or the meaning of the documents due to the absence of a common nomenclature for every PA service.

This lack of field types makes it extremely difficult to develop auto-complete features. However, as shown in the results of the two first usability tests, users are not willing to abdicate from this feature.

Thus, it was decided that Chappie will try to internally map fields that may have the same meaning based on previous values set by the citizen in those fields. In situations where a field

with that name was not filled by the user, Chappie will give suggestions to the user while typing, like a browser does, with the difference that in Chappie, the auto-complete features will not be as powerful as these applications'.

4.3 Usability goals

When in the home screen, it should be easy to the user to identify the options he has available in the application, namely starting a service or adding a new one, viewing the history of previous services or accessing his wallet. These core features should be reachable with the maximum of one click from the home screen.

When performing any given action or moving away from the home screen, it should be clear to the user that he can go back to the previous screen with a click of a button in the top left corner or by using the back button of his device.

When leaving a service after it being started, the user should be presented with a dialogue modal that asks him if he wants to stop the service provision or save the current state.

The user should be prevented from leaving a service during its provisioning. A descriptive message should be shown informing him that he must wait until the provision is finished and the provisioning progress should be made visible.

When in a service, it should be immediately clear to the user if the current service is a root service or part of another service and that service's name.

The user should always be able to go back in his actions in the wizard, except in situations where the application clearly states that doing so will not be possible from that point on.

Regarding performance measures, it should take the user less time to perform an action in Chappie that going through each institution's website and applying for the service there.

The number of steps required for a service provision should be reduced to a minimum that provides an acceptable User Interface that clearly details the involvement of several services in a single session.

4.4 System requirements

This application is meant to be run in tablets, smart-phones and traditional computers. Apart from the disk space required to run it, there should be enough disk space for all the documents the application stores and the ones the citizen adds to it.

Regarding operating systems, this application can be run in Windows 7 or higher, Linux (compatibility depends on the user configuration), MacOS 10.12 or higher, Android 5.0 or newer, and iOS 9 or higher, though its development was only tested in MacOS and Android environments.

The application is distributed bundled with all of its required dependencies, so there is no need for additional applications, except in traditional computers, where a modern browser engine is required.

As for screen size restrictions, the application requires devices whose size of the screens should be of at least 7 inches, and must run in landscape mode. Despite being said the application can run in smart-phones, no tests or adaptations were made to support those devices during the development of the application.

Because the application can run in different environments, some decisions had to be made in order to reduce the amount of work required to make them optimal for each one. For

this reason, more emphasis was put in the tablet interface. This means that certain features that are traditionally available in traditional interfaces, such as hovering an element with the mouse cursor for obtaining more information on that item were not used in this application. Instead, the application relies more in the usage of buttons with more descriptive texts and suitable for interaction with a finger.

Additionally, due to technical challenges described in further sections of this document, the user should also be running a Chappie server in the same network where the citizen application is being executed. This server can run in any operating system that supports the Java Virtual Machine. The technical specifications for this server component and the requirements for its configuration are not described in this section due to them being outside of the scope of this work.

4.5 Platform limitations

Most of the platform limitations were already covered in chapter 3, but there are others that were not addressed in this work due to the amount of work required to implement them not being compatible with the time restrictions of this work. As such, Chappie limitations are as follows:

- Chappie does not support more than one user at a time;
- Chappie does not support the execution of more than one service at a time;
- Chappie can not recover from failures during service provision;
- If one of the services that provide a document for a life event service is unavailable, it is not possible to process the life event;
- The user can not choose what partial service he wishes to obtain obtain first, only after all services are ready can all the services be obtained;
- The citizen application can not run without its server counterpart;
- Autocomplete features are not as complete as the ones provided in web browsers.

During the next sections, the usability tests that were conducted prior to starting developing the application are described. There is also the mention of an alternative user interface that was briefly considered in the initial stages of development.

4.6 First usability test

This first test, conducted with the usage of a low-fidelity prototype[1], focused on testing general user interface organization elements: the services list, the user wallet (at this stage, split in several sections: personal information, documents, and providers), auto-fill, accelerators, and wizard view.

[1]Low-fidelity prototype: A rough representation of a user interface used for finding common errors and validate if the proposed elements, structures and processes are suitable for the application

The list of people who participated in this test is described in section 4.6.1. Section 4.6.2 describes the instructions that were given to the users before starting the test, and sections 4.6.3 to 4.6.7 describe the required steps for each task participants were asked to perform, including the optimal path and tyhe results of each task.

The results of this usability test are described in section 4.6.8.

4.6.1 Participants

Table 4.1 lists the the people who performed the usability test for this first prototype.

In total, 16 people performed the test, 14 of which with ages ranging from 19 to 30 years-old (10 males, 4 females), all students, and two professors with ages of 51 and 53 years-old (one male, one female).

Evaluator ID	Age	Sex	Occupation
1	19	M	Student
2	19	M	Student
3	22	F	Student
4	23	F	Student
5	22	M	Student
6	21	M	Student
7	22	M	Student
8	34	M	Student
9	20	F	Student
10	23	M	Student
11	21	M	Student
12	23	M	Student
13	29	M	Student
14	20	F	Student
15	51	M	Professor
16	53	F	Professor

Table 4.1: Details on the participants of the first usability test

4.6.2 General instructions

During the tests the users were asked to classify each task according to perceived difficulty (easy, medium, hard) and were asked for feedback on what were the tasks they felt more difficulty with and what could be improved.

The following initial story was given to the participants:

"Assume you are a student and your name is David Silva. You want to sign up for a National Students Encounter (for the purposes of this test, it is not relevant what the encounter is).

It is not the first time you are interacting with this application. The application already knows who you are but your taxpayer number is missing from your personal details.

You first want to add your taxpayer information to the application and then you want to sign-up for the encounter.".

Figure 4.4: Chappie initial screen in the first paper prototype

4.6.3 Task 1: Add a personal detail (Taxpayer number)

In this task, users had to add a personal detail (taxpayer number) to their user accounts. This task was meant to check if proposed menus and elements of the main screen were easily understandable by the users.

Expected steps

The user is presented with the initial screen shown in Figure 4.4. In that screen the available services known by Chappie are shown, categorized in sectors.

Before starting signing up for the National Students encounter, the user is expected to select the "People" tab and add his taxpayer information to the already existing account (Figure 4.5).

In order to add this new personal detail, the user has to click the "Novo Item" (New Item) button and fill the two presented boxes with the field name and value. Chappie does not know what type of field that is being inputed nor what fields the user has to add, so both the field name and field value input fields are regular text inputs with no verification on the inserted data (Figure 4.6).

After adding this new field, the user should see the list of fields updated with the newly added value.

After adding his taxpayer information, the user has to return to the initial screen and start his sign-up for the National Students Encounter.

Figure 4.5: "People" screen in the first paper prototype

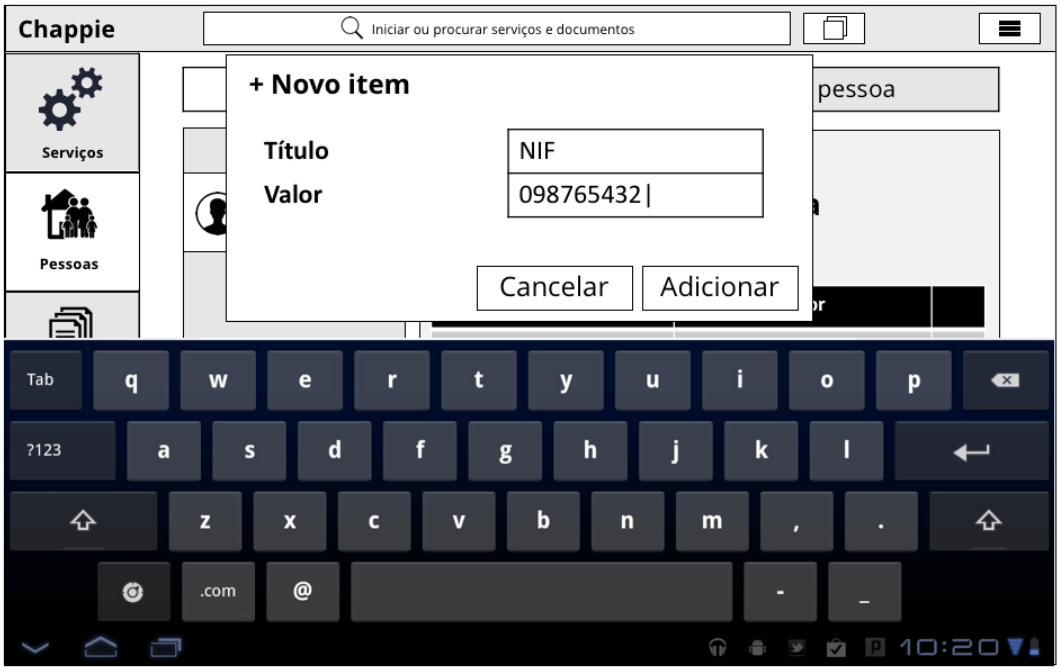

Figure 4.6: New Item dialogue in the first paper prototype

Results from the usability test

Table 4.2 summarizes the results of the usability test for this task.

Evaluator ID	Difficulty	What can be improved	What was more difficult
1	Easy	Fields should be selected from a dropdown	-
2	Easy	-	-
3	Easy	"New Item" should be "New personal detail"	Understanding the difference between services and sectors
4	Easy	-	-
5	Easy	-	-
6	Easy	"New Item" should be "New personal detail"	-
7	Easy	-	Did not understand (s)he had to select "People"; Understanding the difference between services and sectors
8	Easy	"New Item" should be "New personal detail"; Iconography for services should be more clear	Did not understand (s)he had to select "People"
9	Easy	-	-
10	Medium	-	Did not understand (s)he had to select "People"
11	Easy	The application should display the name of the current user	Understanding the difference between services and sectors
12	Easy	Fields should be selected from a dropdown	-
13	Difficult	Fields should be selected from a dropdown	Did not understand (s)he had to select "People"
14	Easy	-	-
15	Easy	"New Item" should be "New personal detail"	Did not understand (s)he had to select "People"; Understanding the difference between services and sectors
16	Easy	-	-

Table 4.2: Results from the first task of the first usability test

Out of the 16 participants, 14 considered the task to be easy, one considered the task hard, and one considered it to be of medium difficulty.

Regarding the steps where users felt more difficulty, six users had trouble understanding they had to select the "Pessoas" (People) option in the menu, and instead either selecting the "Documentos" (Documents) option (evaluators 7, 8, 10, 13, and 15) or the "Preferências" (Preferences) option (evaluator 6).

Regarding suggestions given by the evaluators:

- Evaluators 1, 12, and 13 suggested that Chappie could have a list of predefined fields the users could pick with the option of adding a custom one instead of having to write the field name and value;
- Evaluator 11 suggested that the application should display the name of the current user;
- Evaluators 3, 6, 8, and 15 suggested changing the nomenclature when adding a new field from "Novo Item" (New Item) to "Novo Dado" (New Personal Detail);
- Evaluators 3, 7, 11, and 15 suggested changing the name sectors because in their understanding that nomenclature is not clear enough;
- Evaluator 8 considered the icon used for representing services inadequate.

In the following task, users had to start their sign-up process for the National Students Encounter.

4.6.4 Task 2: Begin a service and obtain its required documents

This task consisted in testing if the concepts of document reusage, circumstances and selection of providers were clear the the users.

For this task, users started where they last left off in the last task and had to start the Sign-up process for the National Students Encounter.

Expected steps

Because in the previous task, the user added a new personal detail, the user starts in that same screen and has to click the "Serviços" (Services) tab to list all available sectors and services.

After that, the user should select the first sector, "Encontro Nacional de Estudantes" (National Students Encounter), view all available services (Figure 4.7) and touch the "Inscrição no Evento" (Sign-up for this event) button. After selecting that event, the user is presented with the screen shown in Figure 4.8.

In this last screen, the user is presented with a welcome screen that introduces him to the service he is about to start.

Before obtaining the list of required documents, the service requires to know a detail on the user (circumstance). In this case, the service has to know if the citizen is a student and therefore can apply to it, or not (Figure 4.9).

If the user answers yes, then the service continues asking additional details. Otherwise, the provisioning stops and no document is obtained (Figure 4.10).

In case the user is a student, the provisioning continues and lists the required documents the user has to provide to the service so he can proceed with the sign-up process (Figure 4.11).

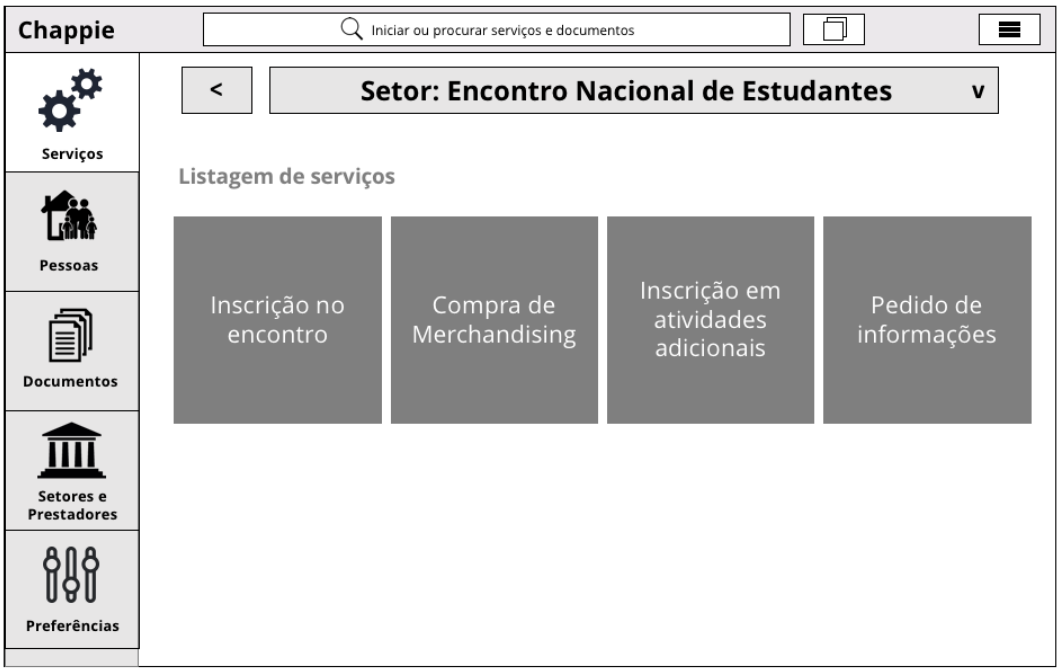

Figure 4.7: Screen showing the list of services available in the "Encontro Nacional de Estudantes" sector.

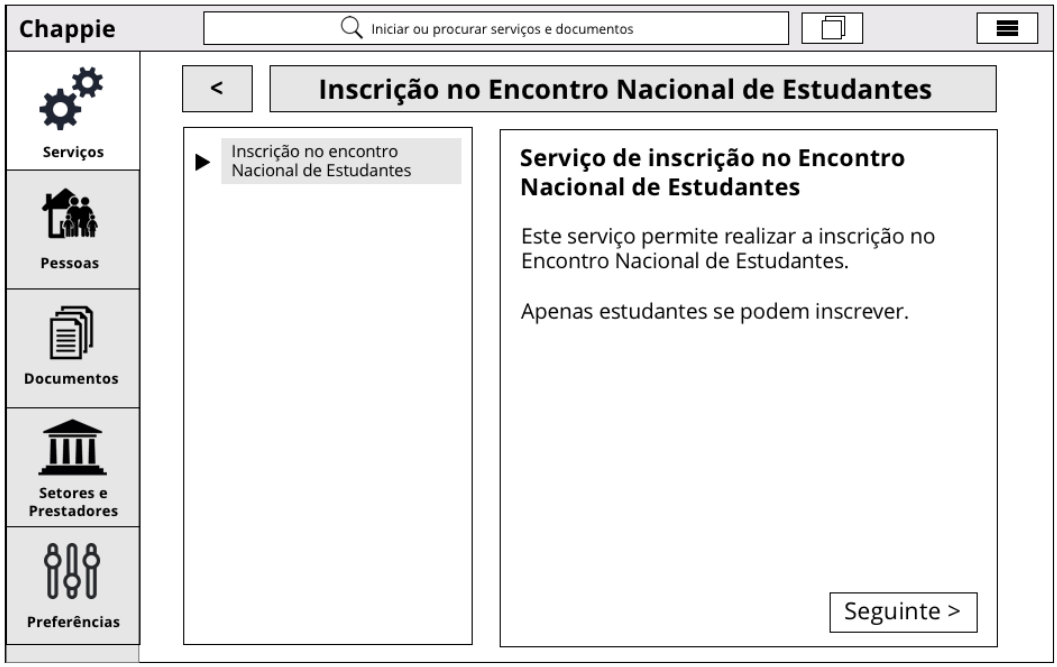

Figure 4.8: Chappie screen to start the National Students Encounter sign-up service.

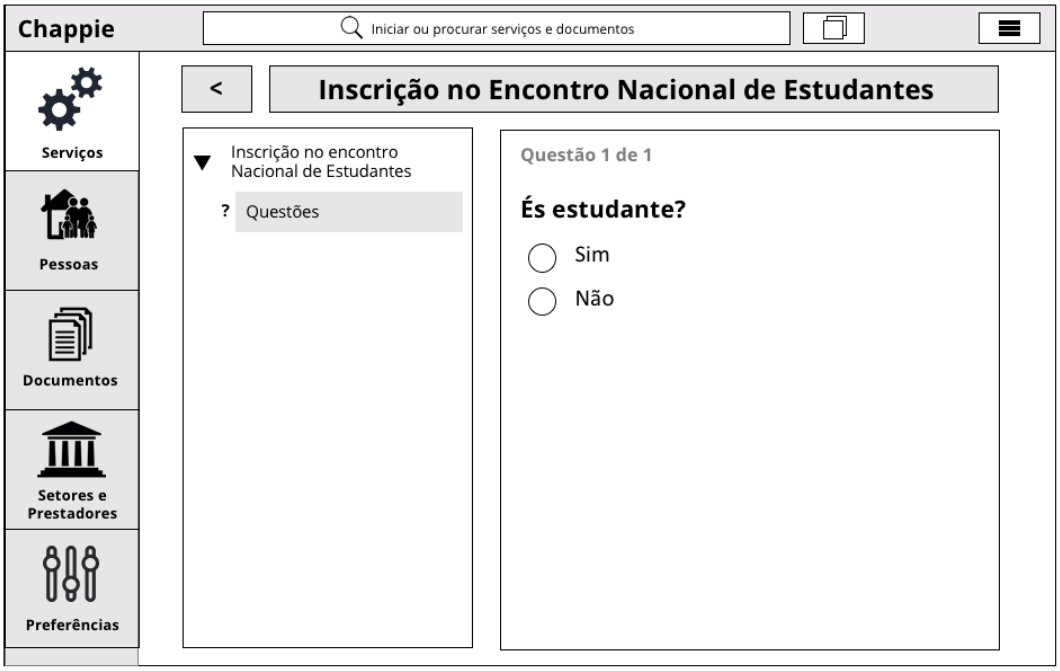

Figure 4.9: A circumstance in the first paper prototype.

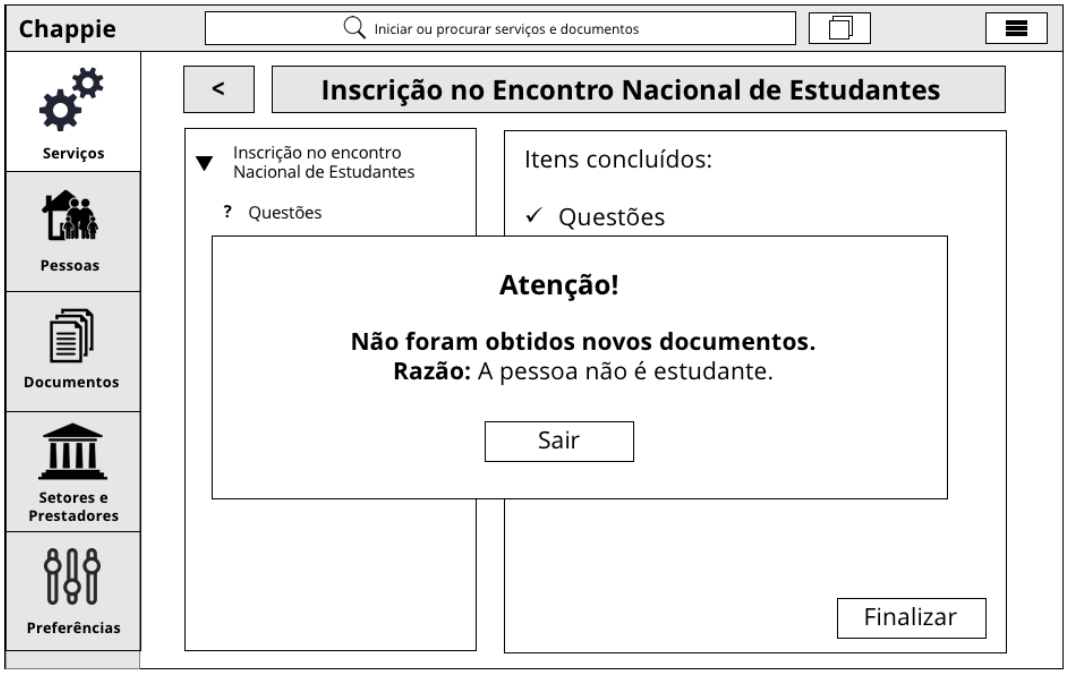

Figure 4.10: Error message shown if the user is not a student

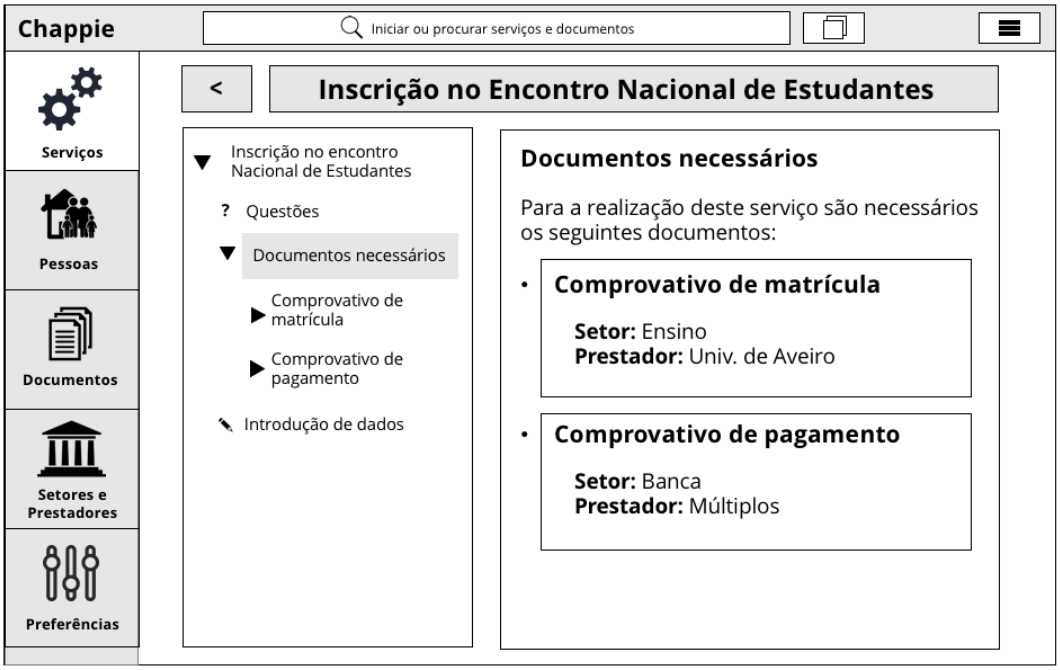

Figure 4.11: Enumeration of required documents for the service

---

Because in Chappie services require other documents that can be obtained from the citizen wallet or through other partial services, the user interface explores those options by presenting the users the screens shown in figures 4.12 and 4.13.

When there is no need for additional interaction, the document can be immediately obtained and the user is shown the message on Figure 4.14.

After all required documents are obtained or provided, the user is asked to provide his sign-up information. That process is covered in the next task.

Results from the usability test

Out of the 16 participants, 13 considered the task to be easy and 3 considered it to having a medium difficulty, as can be seen by the results shown in table 4.3.

Regarding the difficulties the users felt when transitioning between required documents, 9 users did not realized they were obtaining another document when transitioning from "Comprovativo de Matrícula" to "Comprovativo de pagamento", while 2 users said that was the behavior they were expecting from the application.

One other user suggested that the way Chappie presents that a payment has to be made (by asking for a proof of payment ("Comprovativo de pagamento") is not clear enough.

When concerning documents storage, 4 of the users said it is not clear that a document that is obtained is stored in the citizen's wallet, and another even suggested a notification informing the user that the document was stored there.

In regards to how Chappie handles service provision, 4 of the users thought the documents obtained were immediately sent to the service that was requesting them, failing to understand they were being stored locally.

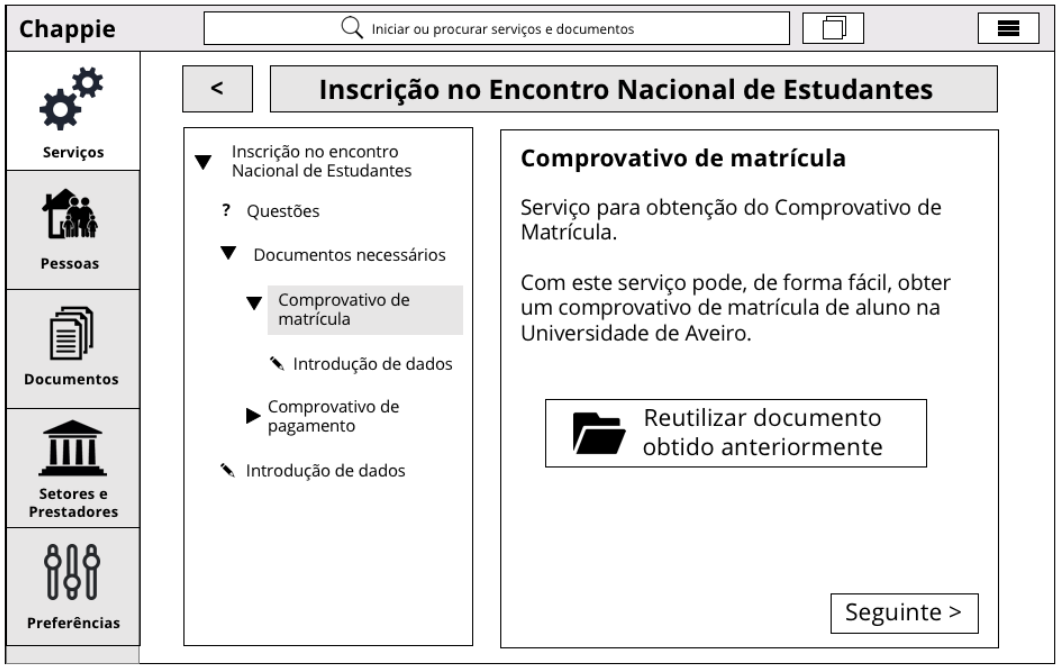

Figure 4.12: Screen showing the possibility of document reusage

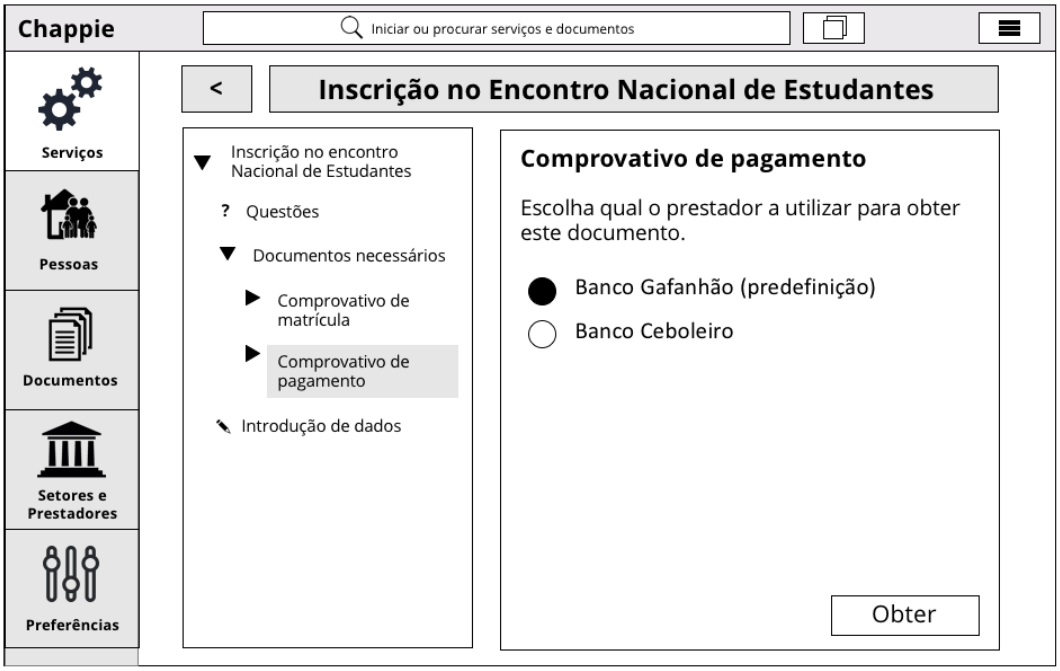

Figure 4.13: Select provider screen of the first paper prototype

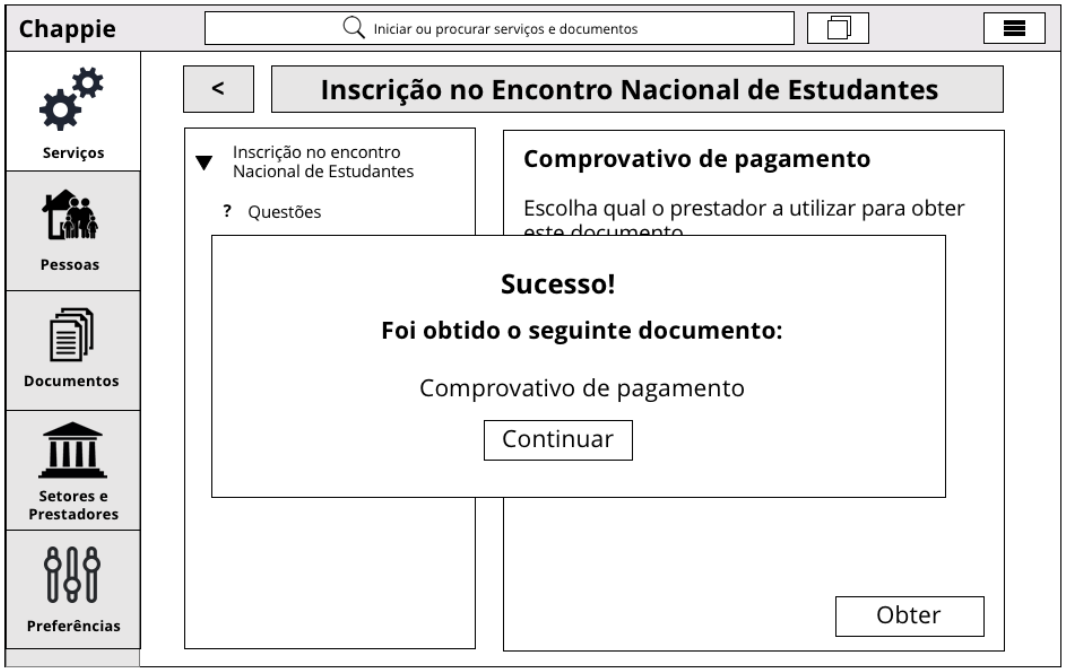

Figure 4.14: Message shown after a document being obtained

In what concerns navigation, one of the users had difficulties going back to the previous step, while another completely abandoned the service entirely without the system informing him about what he was doing.

4.6.5 Task 3: Fill-in input fields

In this tasks the users were asked to fill the input fields that were being presented to them.

This task was useful to understand how users acted when presented with a helper for form filling. It was also useful to test if users were more willing to use this helper, traditional keyboard input methods or keyboard accelerators.

Expected steps

When presented with a form (Figure 4.15), users were expected to input values using one of three ways: inputting the value directly — taking advantage (or not) of an auto-filler -, selecting a value from a list of know values, or with the usage of keyboard accelerators (shown in this test only for checking if the evaluators could understand it).

Figures 4.16 and 4.17 show the behavior of the application when using the filler helper. If the user chose to use the keyboard for adding values, the auto-fill helper would be displayed as well as the keyboard accelerators available for that field (Figure 4.18).

After filling all the fields, the user is presented with a confirmation screen saying the service is ready to be obtained.

Evaluator ID	Difficulty	What can be improved	What was more difficult
1	Easy	Did not understand the difference between stored documents and obtained documents	Transition between documents was too fast
2	Easy	-	-
3	Medium	-	Transition between documents was too fast
4	Easy	-	Transition between documents was too fast; Had troubles going back
5	Easy	-	Transition between documents was too fast
6	Easy	-	Transition between documents was too fast
7	Easy	-	Transition between documents was too fast
8	Easy	-	Did not understand the difference between stored documents and obtained documents
9	Easy	It should be more clear that a new document is going to be obtained	Transition between documents was too fast
10	Easy	Keyboard accelerators are confusing	-
11	Easy	The application should indicate that a document was obtained	Transition between documents was too fast
12	Easy	It should be more clear that a new document is going to be obtained	-
13	Easy	It should be more clear that a new document is going to be obtained	-
14	Medium	-	Transition between documents was too fast
15	Medium	It should be more clear that a new document is going to be obtained	Transition between documents was too fast
16	Easy	-	The user unintentionally abandoned the service

Table 4.3: Results from the second task of the first usability test

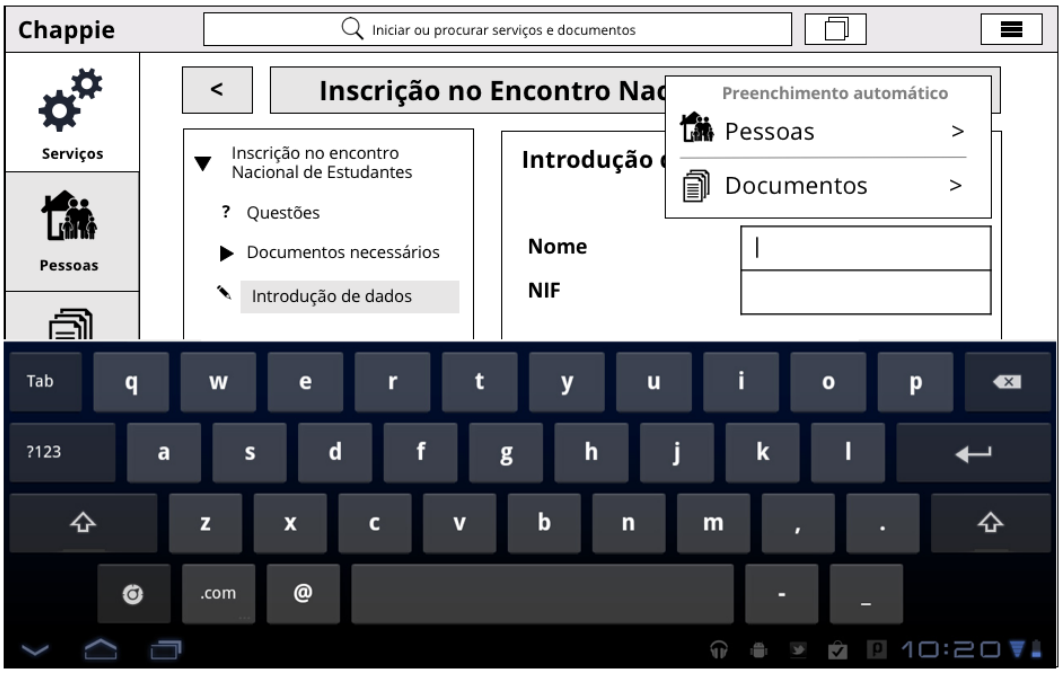

Figure 4.15: Options shown to the user when inputing a field.

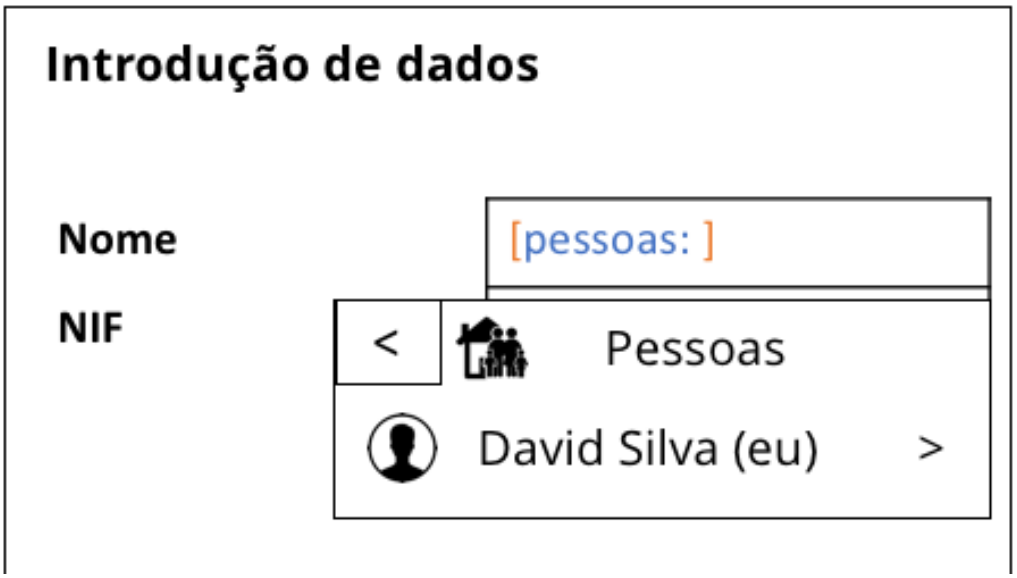

Figure 4.16: Menu for selecting a field value from a folder

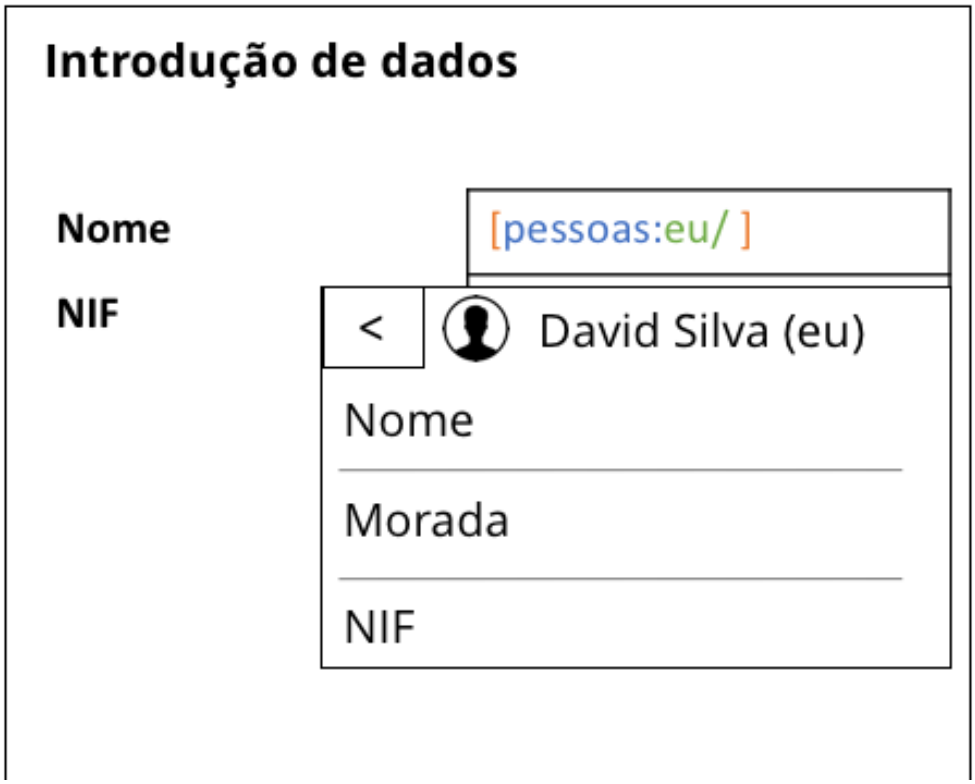

Figure 4.17: Available auto-fill items presented in the "Eu" (Me) folder

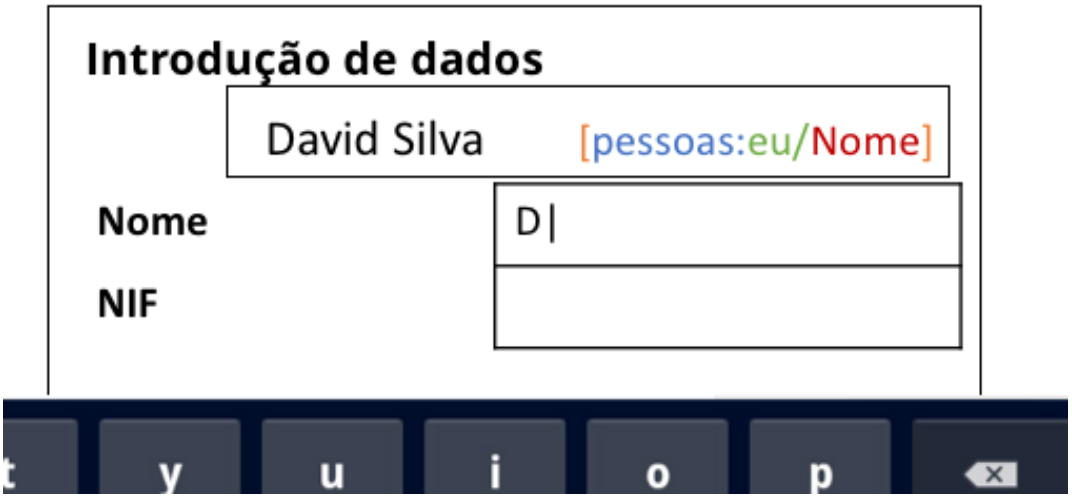

Figure 4.18: Auto-fill screen with keyboard accelerators displayed on the right

Results from the usability test

Results are shown in table 4.4.

Evaluator	Input method	Difficulty	What can be improved	What was more difficult
1	Assisted Input	Easy	Remove the need for selecting "Person" and "me"	-
2	Keyboard Input	Easy	-	-
3	Assisted Input	Easy	-	-
4	Assisted Input	Easy	-	Did not understand the taxpayer number is in documents
5	Assisted Input	Easy	Remove the need for selecting "Person" and "me"	-
6	Keyboard Input	Easy	-	-
7	Keyboard Input	Easy	-	Keyboard accelerators are confusing
8	Keyboard Input	Easy	-	Keyboard accelerators are confusing
9	Keyboard Input	Easy	-	Keyboard accelerators are confusing
10	Keyboard Input	Easy	-	-
11	Keyboard Input	Easy	Fields should be filled automatically	-
12	Keyboard Input	Easy	Fields should be filled automatically	Keyboard accelerators are confusing
13	Keyboard Input	Easy	Fields should be filled automatically	-
14	Assisted Input	Easy	Selecting one item should auto-fill the others	-
15	Keyboard Input	Medium	Data input should come before the required documents; Fields should be filled automatically	Did not understand (s)he was about to request a new document; Keyboard accelerators are confusing
16	Assisted Input	Medium	Fields should be filled automatically	Keyboard accelerators are confusing

Table 4.4: Results from the third task of the first usability test

From the evaluators, 14 considered this task to be easy, and 2 considered it of medium difficulty.

When presented with the form for inserting information, 5 of the evaluators expected them to be auto-filled already with the information they provided in the first step.

Regarding the way the users preferred to interact with the system, the majority of the evaluators (10) immediately looked at the keyboard and started typing the information that was being requested. All of the evaluators who used the keyboard also used the auto-complete feature, and one of them suggested that when filling one field using the auto-complete, the other fields should be filled automatically too.

Only 6 evaluators used the assisted input method, with 2 of them suggesting the removal of having to select "Pessoas" and "David Silva (eu)" to view the list of available field values, and another incorrectly selecting the "Documents" option.

When asked about the keyboard accelerators, 5 evaluators said they were confused by the feature and did not understand it, 2 said they understood the feature and may use it in the future, and the remaining 9 evaluators said that although they understood what it was they would not have used them if they were available.

Additionally, one of the evaluators suggested that the data input should be made before adding the required documents.

4.6.6 Task 4: Understand that the service was finished

This tasks merely consisted on presenting a commit page where the user could review all the data that he provided and confirm he is ready for finishing the service.

Expected stepts

Figure 4.19 shows the page that is shown to the user when the service is ready to be obtained.

It is expected that the user clicks "Finalizar" (Finish) or go back to verify if the information is correct.

After clicking Finish, the user is presented with the screen shown on figure 4.20.

Results from the usability test

All the 16 users could perform this task with no issues, as can be seen in the results presented in table 4.5.

4.6.7 Task 5: Switch Sectors

This was another simple test used to check if the concept of sectors was well understood and if it made sense to give the users the ability to switch sectors using a dropdown menu.

Expected stepts

The users are expected to go to the services page and open one of the sectors that are present to them. Having that sector open, they should be able to see that the title bar is clickable and either click it or the dropdown on the right and select a new sector (Figure 4.21).

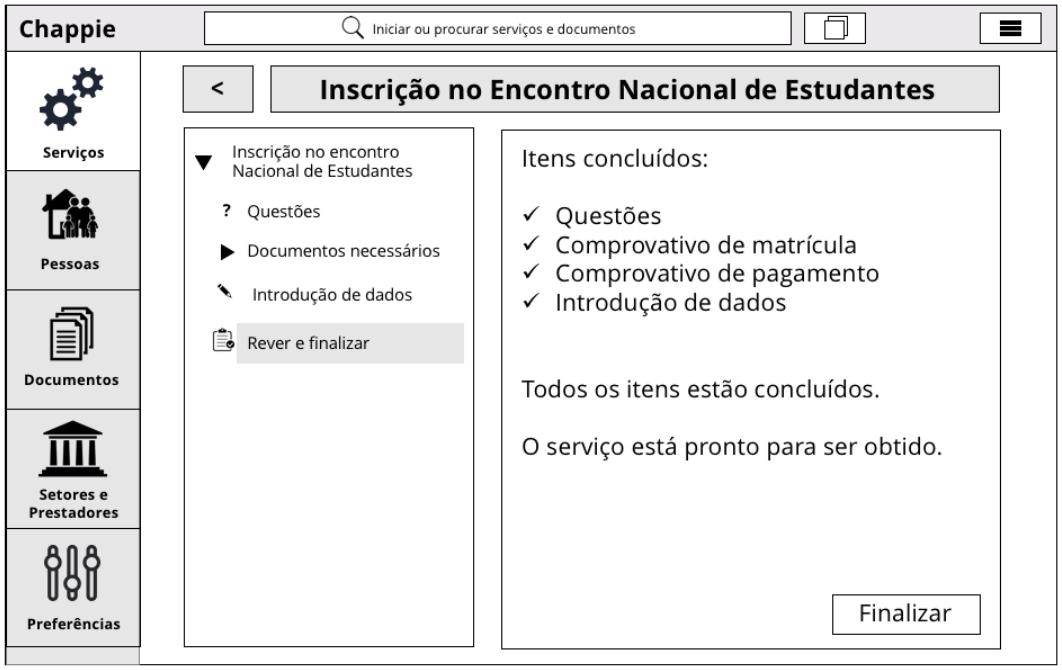

Figure 4.19: Review service screen

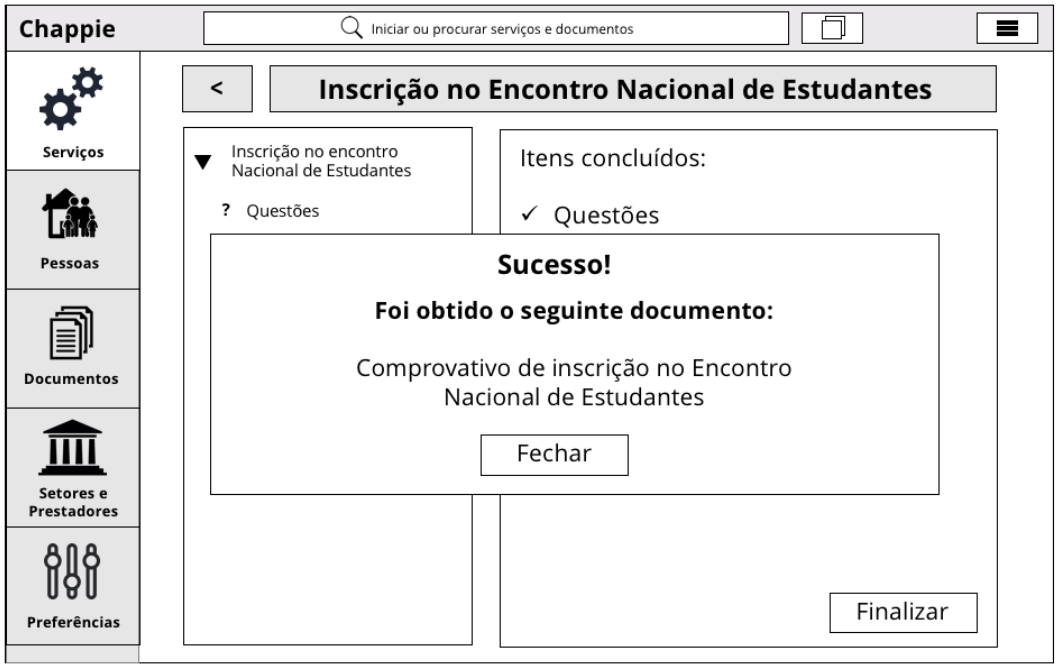

Figure 4.20: Success message displayed when the user finishes the service

Figure 4.21: Change sectors drop-down

Results from the usability test

Results in table 4.5. Of the 16 evaluators, only 4 were able to complete the task with success, which lead to the option being removed in future approaches.

4.6.8 Conclusions

By analyzing the results from the usability test, it can be concluded that the menu structure that was initially planned is not suitable for the application. Users were confused with the amount of options that they were given, and, as such, this was a problem that we addressed in the second paper prototype. It is also clear that there are both nomenclature and iconography problems that need to be addressed in future approaches.

Regarding the way the service provision screens are presented, the test results show that there is still room for improvement, specially in how messages are displayed, transitions from documents are handled, and errors are prevented.

As for the suggestions of Chappie knowing some personal identification fields and even auto-completing forms with them, that feature was not initially planned. Fields in the CHA-PAS Model do not have types and can have the same meaning with different names ("Full name" and "Name" are fields with different names that have the same meaning). However, because both an auto-fill was an highly expected feature, a model of guessing the value of a field will be applied based on previous interactions with fields with the same name.

Another conclusion that can be taken from the test results is that the existence of an helper for filling the fields did not met the expectations of the users, who were expecting different features more common in other application. Keyboard accelerators were also not well received.

Evaluator ID	Task 4	Task 5
1	Success	Failed
2	Success	Failed
3	Success	Failed
4	Success	Failed
5	Success	Success
6	Success	Failed
7	Success	Success
8	Success	Failed
9	Success	Success
10	Success	Failed
11	Success	Failed
12	Success	Success
13	Success	Failed
14	Success	Failed
15	Success	Failed
16	Success	Failed

Table 4.5: Success rate of the fourth and fifth task of the first usability test

In conclusion, both the user interface and the service wizard were well received by the users, apart from the problems mentioned above. However, the overall satisfaction may be also related with the number of steps required for applying for a service being so small.

Because of these conclusions, a new usability test in an improved application was carried some months later. The new prototype is described in the next section.

4.7 Second paper prototype

After applying some of the suggested changes from the previous usability test, a new experience was conducted with a substantially bigger service. The reasons behind this new test were trying to understand if the right path was chosen regarding the changes that were applied to the menu system, test if the interface was still suitable for a bigger service than the one used in the previous test, and also introduce new concepts. Among those new concepts are a skippable on-boarding experience and a new home screen featuring a search box.

Functionalities that were not very used or not very well received were removed such as the filler helper.

Support for predefined field types and auto-fill of forms were also among the new features added to the application.

4.7.1 Participants

The usability test for this prototype was conducted with a sample of 10 people, 8 of which with ages raging from 19 to 25 years-old (7 males, 1 female), all students, and two professors with 51 years-old (2 males). Table 4.6 shows the detailed list of evaluators.

Evaluator ID	Age	Sex	Occupation
1	23	M	Student
2	22	M	Student
3	24	M	Student
4	21	M	Student
5	19	M	Student
6	24	M	Student
7	51	M	Professor
8	51	M	Professor
9	22	F	Student
10	25	M	Student

Table 4.6: Details on the people who participated in the second usability test

4.7.2 General instructions

Before starting the test, users were given the following introduction:

"Assume you are a student and your name is David Silva. You lost your student card and you want to use Chappie to recover it. It is the first time you are using Chappie, but one of your friends who used the application already said that you can add all your personal information before starting a service and after that the process will be done almost automatically".

Users were also given two forms to fill-in: one where they could write their perceived difficulty in each of the tasks in a scale from 1 to 5 (1 being "very hard" and 5 being "very easy"), and another where they could give their opinion regarding their experience with the system by classifying several sentences in a scale from 1 to 5 (where 1 means totally disagree and 5 means totally agree) and by adding additional comments they may find relevant.

For each task, it was measured the time the users took to perform it as well and notes were taken regarding the user performance. The number of clicks was not measured, but situations where the users followed a path different from the optimal one were registered.

4.7.3 Task 1: Add a personal detail (Taxpayer number)

This task served the purpose of testing the new on-boarding experience. These screens allowed the user to create a new user account and add there their personal information in the form of personal cards.

Expected steps

When starting the application for the first time, the user is immediately greeted with the on-board screen that explains what the application is and helps him/her set up his/hers new account (Figure 4.22).

The user only had to click next at this step to start the process of creating a new profile, as shown in figure 4.23. In this screen, the users are informed on the benefits of having a local account in Chappie.

When at this screen, users are expected to select the option to create a new profile, which leads them to a new screen where they can fill in some of their personal details to be added to the account (Figure 4.24).

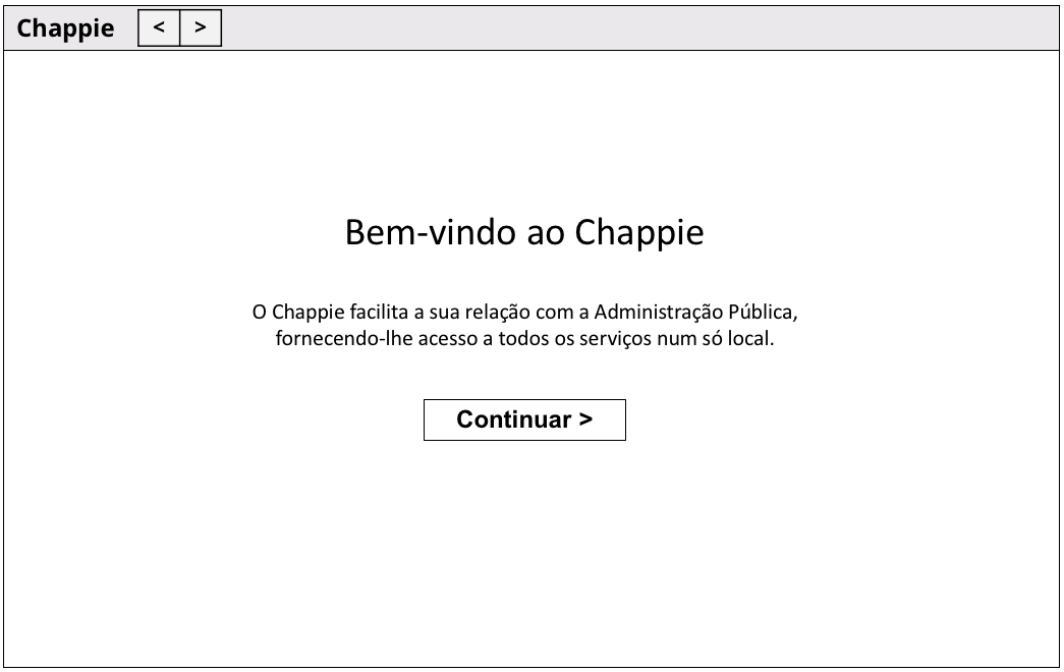

Figure 4.22: Chappie on-boarding welcome screen

Figure 4.23: Create Profile Screen

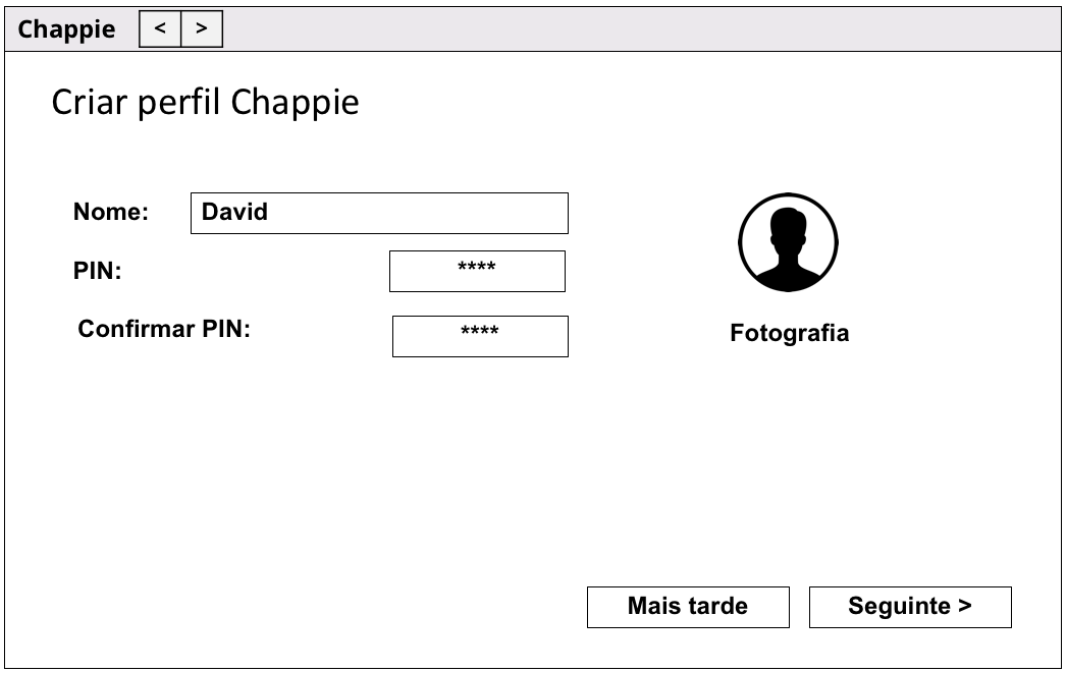

Figure 4.24: User Profile Form

After filling in all their personal details, the users are asked if they want to create a new Chappie identity card with their personal information. These cards serve as a representation for fields that can be aggregated under a same group. In the screen shown in figure 4.25, users are able to add Citizen cards or student cards. They are expected to create both, one for personal identification and another for applying for the service that is being studied in this test.

When adding a card, users can select if they wish to insert the card data manually (Figure 4.26) or import it from a smart card reader (Figure 4.27). Users could also add additional fields to the cards (Figure 4.28).

Whenever a new document was created, users were given visual indication about it (Figure 4.29).

After all the process was concluded, users were shown the screen in figure 4.30.

4.7.4 Usability test's results

All the users were successful with the task and considered it very easy (5/5). The average time for this task was 1 minute and 17 seconds, with a minimal value of 40 seconds and maximum value of 1 minute and 50 seconds, as can be seen in the values presented in table 4.7.

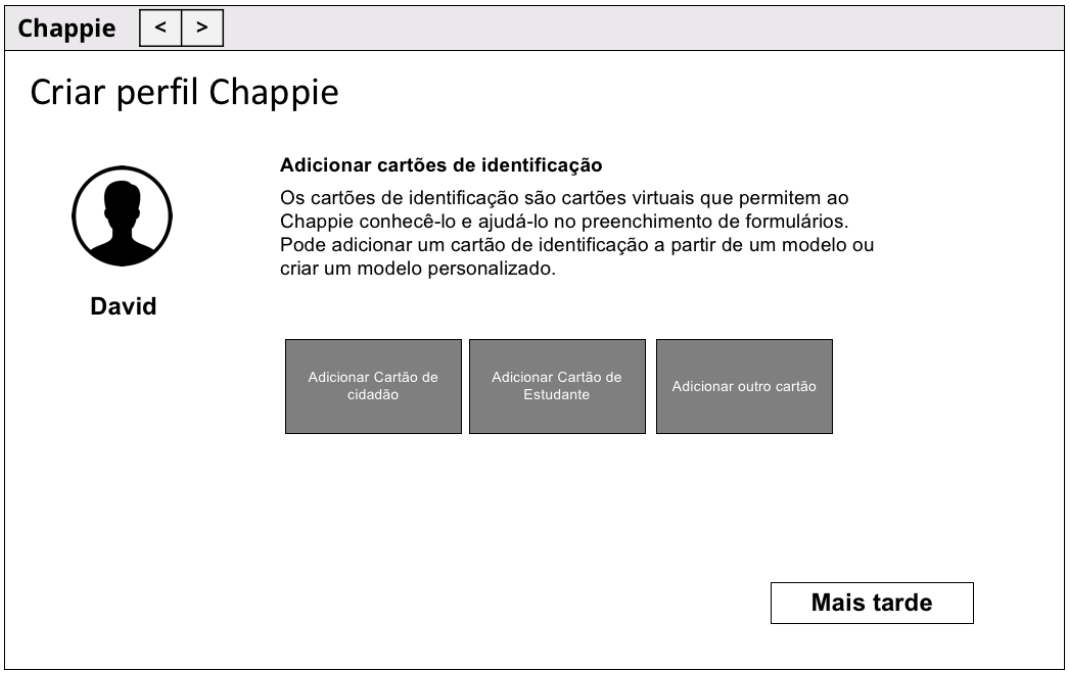

Figure 4.25: Add identity cards screen

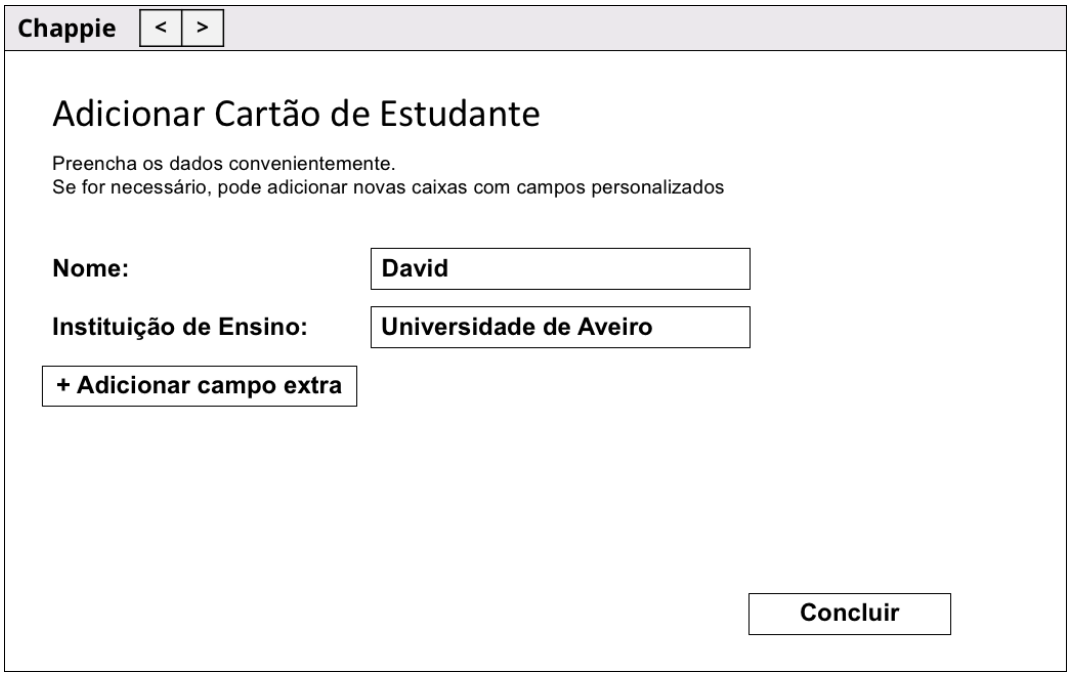

Figure 4.26: Manual insertion of field values in an Identity Card.

Figure 4.27: Chappie importing data from a smartcard

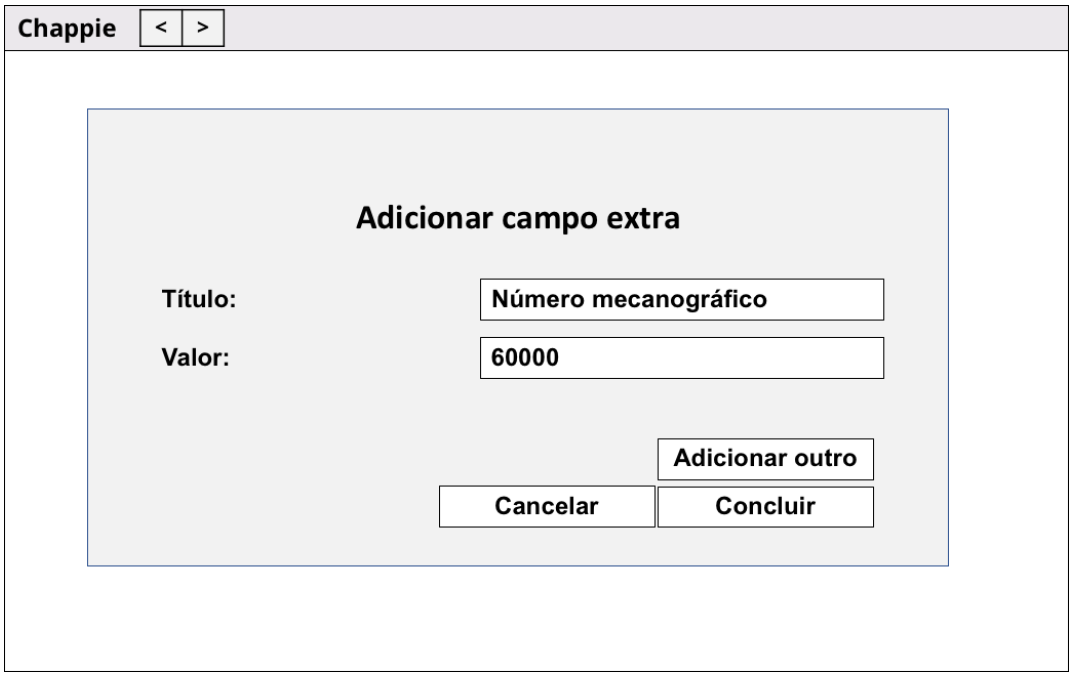

Figure 4.28: Adding a custom field to an Identity Card.

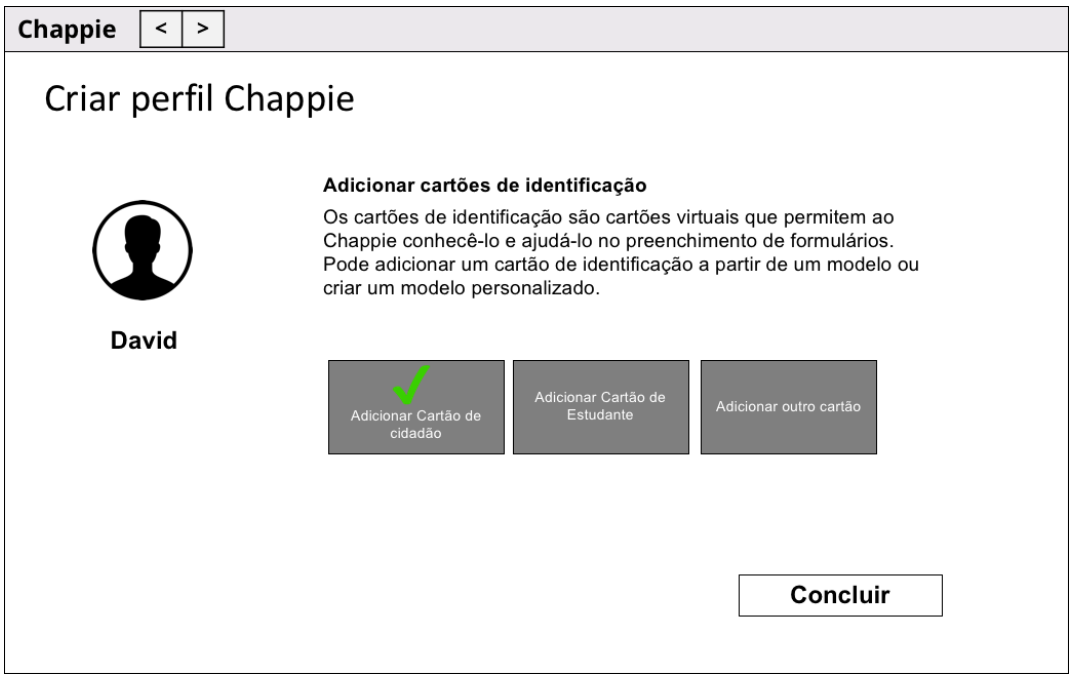

Figure 4.29: Indicator used in an already obtained card.

Figure 4.30: On-boarding finish screen.

Evaluator	Completed	Time	Made mistakes	Felt lost	Asked for help
1	Yes	00:40	No	No	No
2	Yes	00:40	No	No	No
3	Yes	01:30	No	No	No
4	Yes	01:50	No	Partially	No
5	Yes	01:20	No	No	No
6	Yes	01:10	No	No	No
7	Yes	01:30	No	No	No
8	Yes	01:16	No	No	No
9	Yes	01:10	No	No	No
10	Yes	01:49	No	Partially	No

Table 4.7: Observer's notes for the first task of the second usability test

Evaluator ID	Reported ease-of-use	Observed ease-of-use	Observations
1	5	5	Was going to cancel the reading of the citizen card
2	5	5	Was going to cancel the reading of the citizen card
3	5	5	-
4	5	4	Was going to cancel the reading of the citizen card
5	5	5	Was going to cancel the reading of the citizen card
6	5	5	-
7	5	5	-
8	5	5	-
9	5	5	Was going to cancel the reading of the citizen card
10	5	5	Was going to cancel the reading of the citizen card

Table 4.8: Reported and observed difficulty of the first task of the second usability test

The difficulty of the task was rated 5 out of 5 in terms of ease of use. None of the users made any mistake but two of them felt a bit lost in the process. Out of the 10 evaluators, 6 of them were about to click the Cancel button shown in figure 4.27, as can be seen in the results displayed in table 4.8. However, all of them realized what they were about to do without the

Figure 4.31: Chappie log-in screen

need for intervention from the observer.

4.7.5 Task 2: Find the required service

This task served the purpose of understanding if the users were more willing of using a search box to find a situation they are in need or if more menu-based approaches were preferred.

Expected steps

Having a new account already created, users are now expected to log-in to that account and find the service they are interested in applying to. Figures 4.31 and 4.32 display the steps required to log in to the application.

After successfully logging in, evaluators have to search for the service they want to apply. This can be done using two approaches: by using the search box that appears immediately after signing in (Figures 4.33 and 4.34) or by scrolling to a more traditional view (Figures 4.35 and 4.36).

Services can be accessed either by selecting a life event or the requested service directly. If an event is chosen, the event description page (Figure 4.37) is shown. From there, the user is then able to start the service he is in need.

4.7.6 Usability test's results

The time the users spent in this task averages to 33 seconds, with a minimum of 10 seconds and the maximum of one minute, as can be seen in table 4.9.

Figure 4.32: Chappie PIN input screen

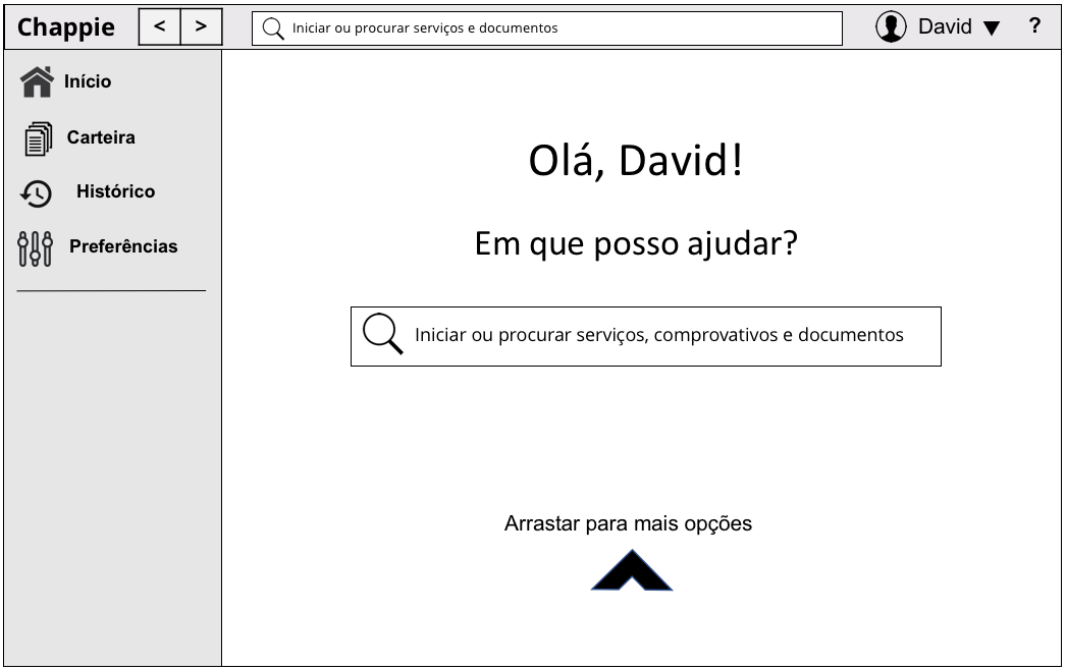

Figure 4.33: Chappie search screen

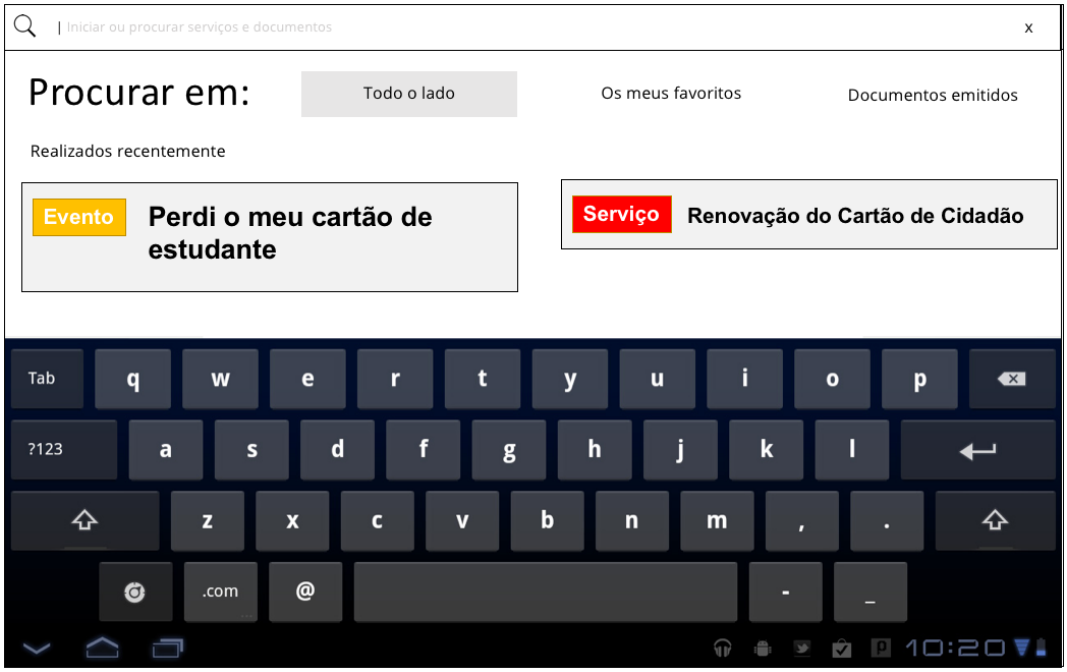

Figure 4.34: Chappie search results screen

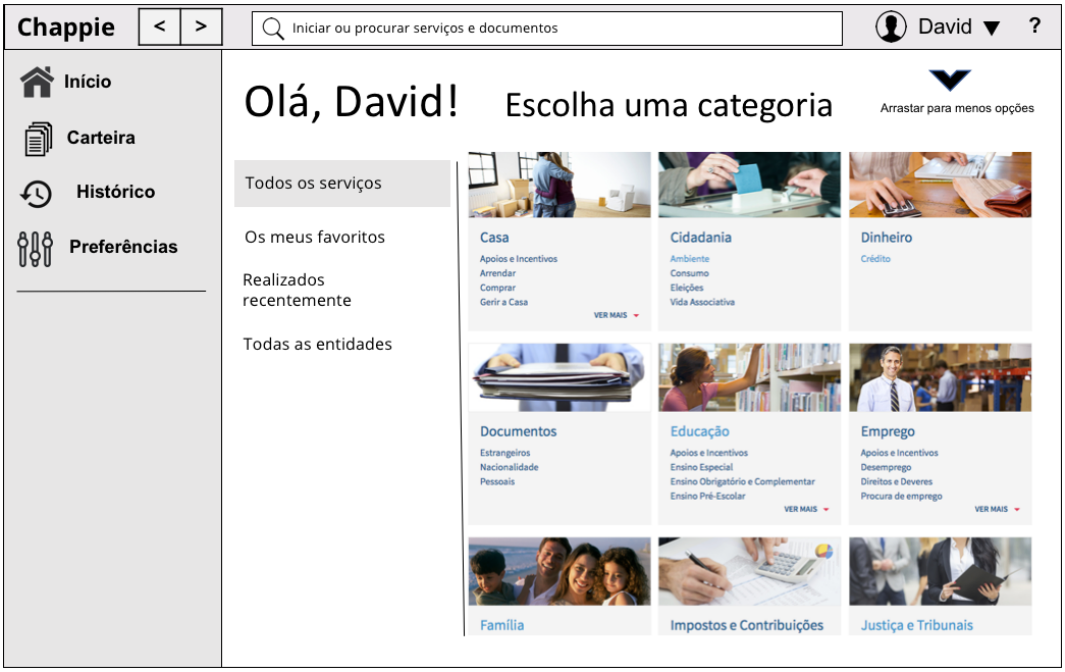

Figure 4.35: List of services shown in the second prototype

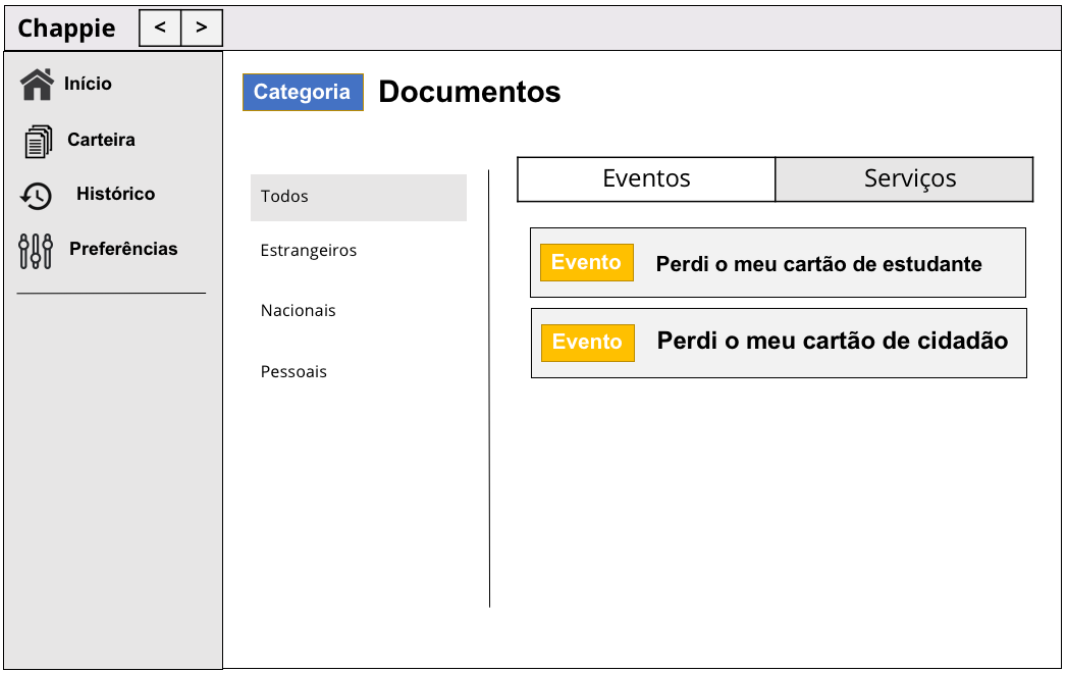

Figure 4.36: List of services available in the "Documents" category

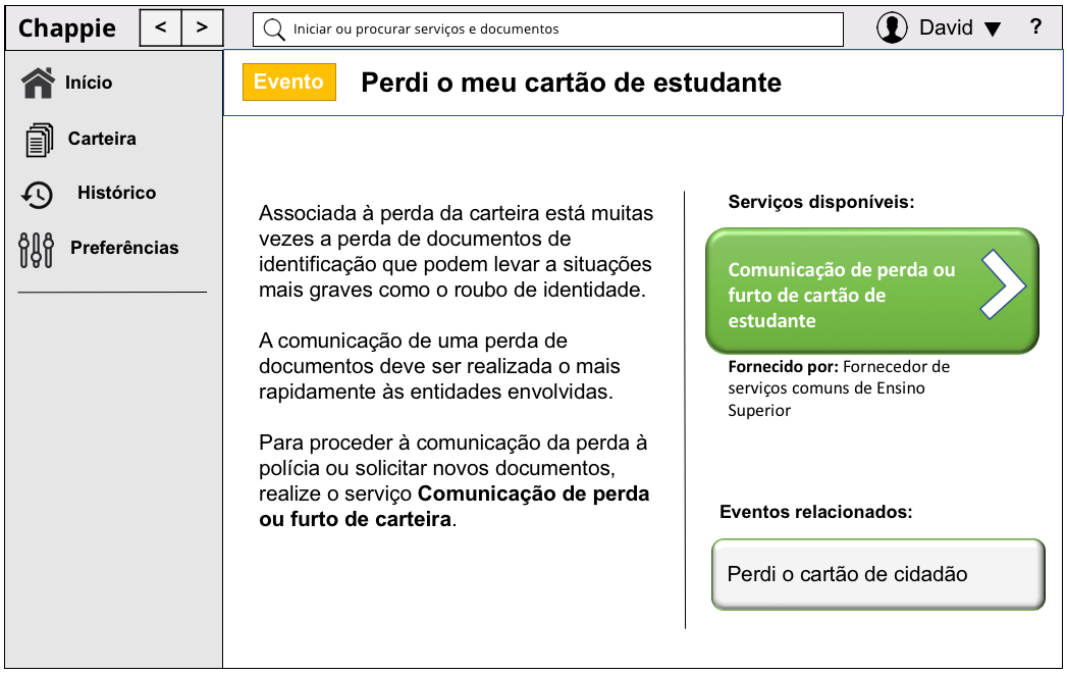

Figure 4.37: Event description page

Evaluator ID	Completed the task	Time	Made mistakes	Felt lost	Asked for help
1	Yes	00:10	No	No	No
2	Yes	00:10	No	No	No
3	Yes	01:00	No	No	No
4	Yes	00:50	No	Partially	No
5	Yes	00:36	No	No	No
6	Yes	00:46	No	No	No
7	Yes	00:20	No	No	No
8	Yes	00:20	No	No	No
9	Yes	00:20	No	No	No
10	Yes	01:00	Some	Partially	Yes

Table 4.9: Observer's notes for the second task of the second usability test.

This task was rated 4,8 out of 5 in terms of ease of use. All of the users could complete the task with no problems, except for one of the users who requested for help immediately in the first screen due to difficulties in finding the traditional menu-based approach.

Evaluator ID	Reported ease-of-use	Observed ease-of-use	Observations
1	5	5	Used the search field
2	5	5	Used the search field
3	5	5	Used the menu system
4	5	4	Used the menu system
5	5	5	Used the menu system
6	4	5	Used the menu system
7	5	5	Used the search field
8	5	5	Used the search field
9	5	5	Used the search field
10	4	4	Used the menu system; Requested help to find the menu system.

Table 4.10: Reported and observed difficulty of the second task of the second usability test.

As for the preferred method for finding services in Chappie, the same number of users used the search box and the menu system, as can be seen in the results from table 4.10.

4.7.7 Task 3: Fill-in the information requested by the service

This task consisted in presenting the users a service which required more steps than the one used in the previous usability test. This task was also used to demonstrate the ability of Chappie for auto-filling the fields with information it had previously gathered from the user.

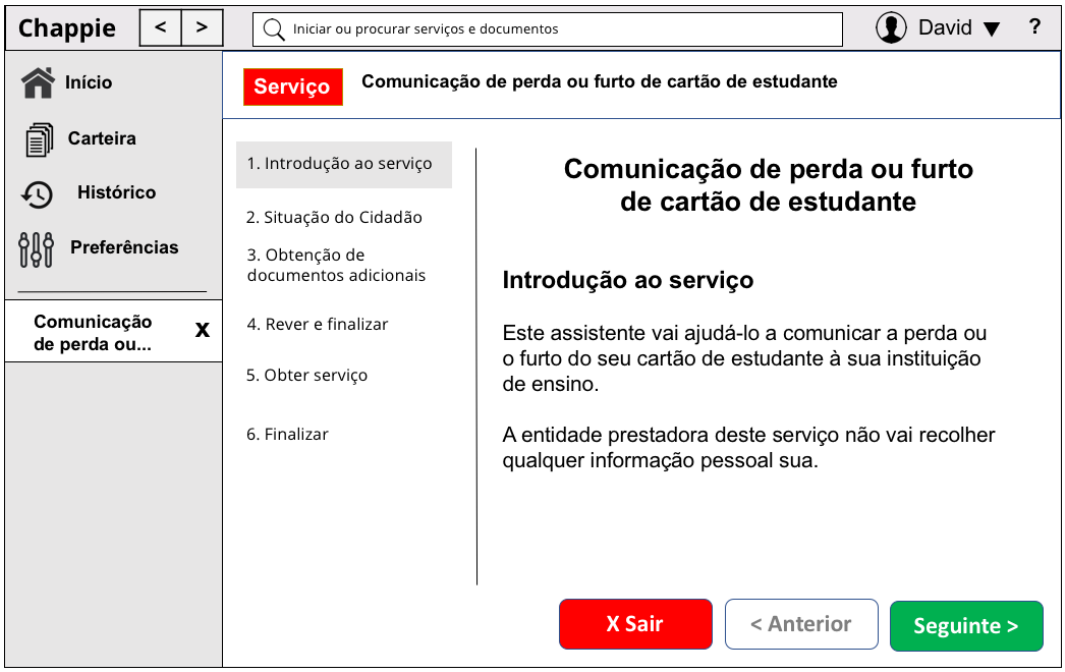

Figure 4.38: Welcome screen for a service in second paper prototype

---

Expected steps

After selecting the option to start the service from figure 4.37, evaluators were presented with the initial screen for the service they were applying to (Figure 4.38).

After skipping this screen, evaluators were presented with another screen where they had to select the provider they wanted to use for obtaining a given document (Figure 4.39).

After selecting the provider, users are presented with the list of the documents that the service requires them to provide (Figure 4.40).

Users have the option of clicking "Next" and continuing with the first document on the list, or select the document they wanted to obtain (in this case, it is only one). After selecting the document, the partial service that issues that document is started and the users are presented with a welcome screen for that service (Figure 4.41).

Partial services, like root services, can require the users to answer a given circumstance (Figure 4.42), input data (Figure 4.43) or obtain additional documents from other services.

Before moving to the next required document and after all documents provided in partial services are obtained, Chappie displays a review screen for the service, where the user is able to view the information he provided either through input fields or documents (Figure 4.44).

4.7.8 Usability test's results

The results of the usability test for this task are described in tables 4.11 and 4.12. The average time the evaluators took for completing the task was of 1 minute and 41 seconds, with the highest time of 2 minutes and 50 seconds and the lowest of 1 minute. The task was rated 4,8 out of 5 in terms of ease of use by the evaluators.

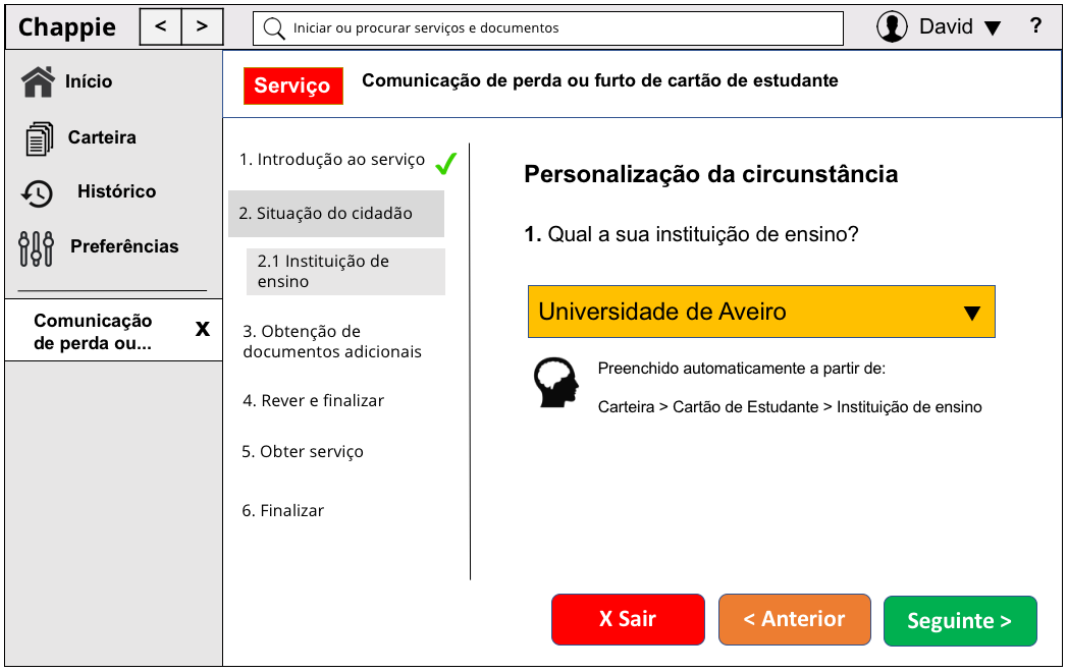

Figure 4.39: Provider selection screen in second paper prototype

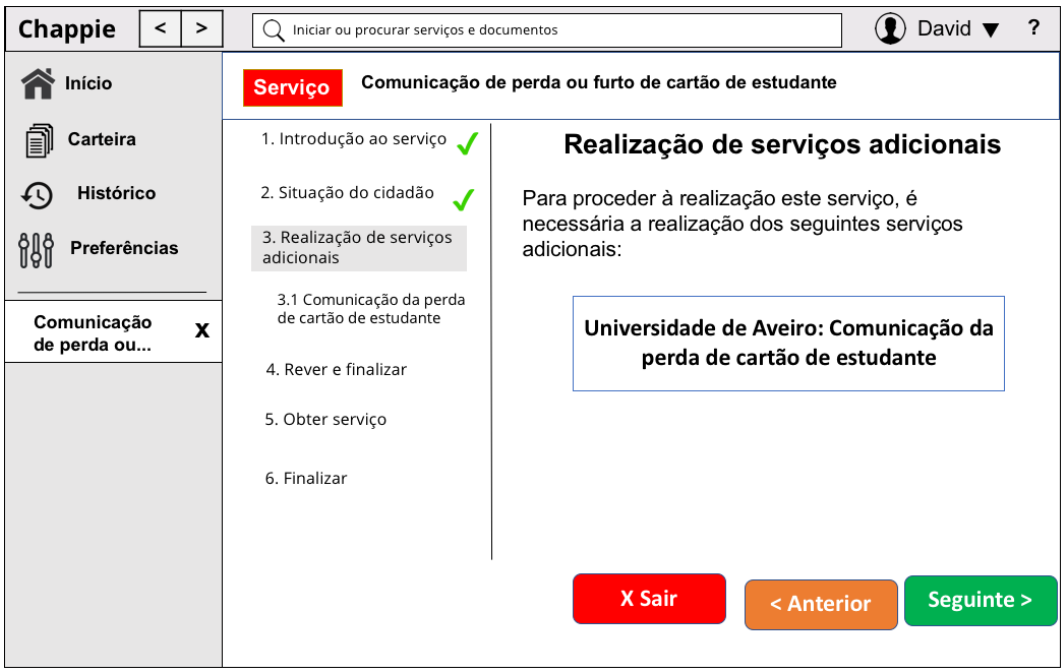

Figure 4.40: Required partial services for obtaining a document

Figure 4.41: Welcome screen for a partial service in second paper prototype

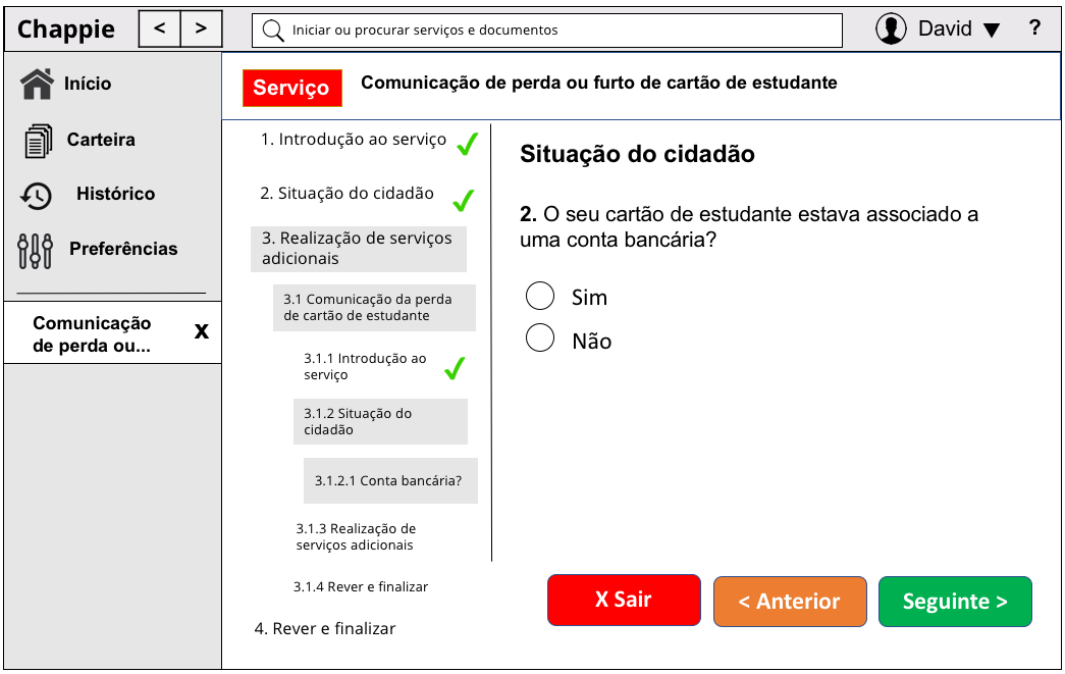

Figure 4.42: Circumstance of a partial service in the second paper prototype

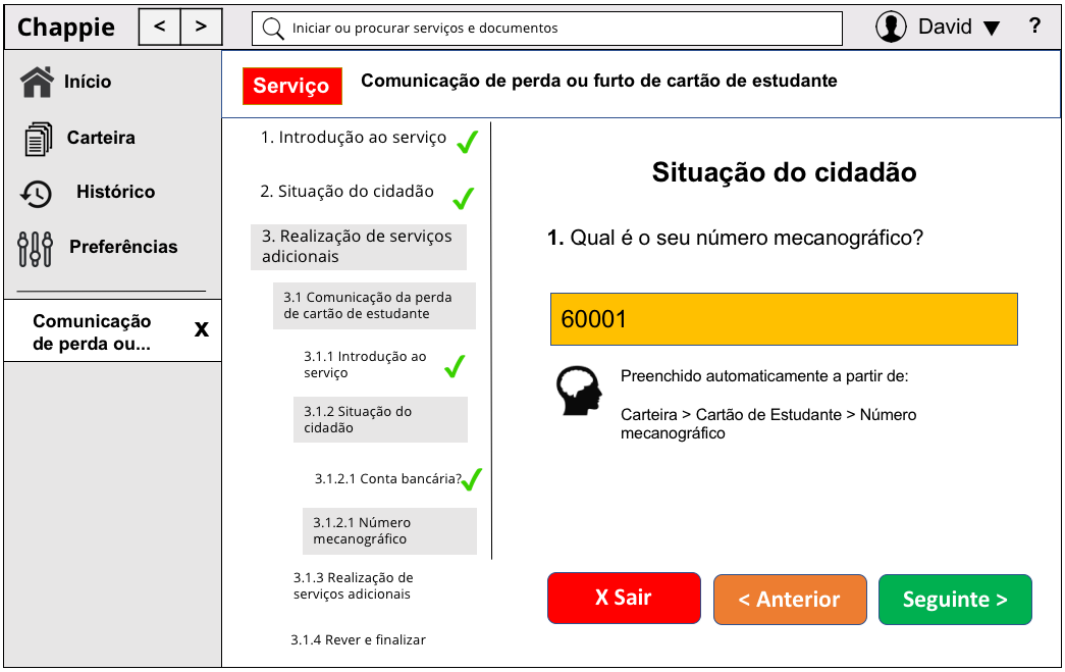

Figure 4.43: Data input required for a partial service in the second paper prototype

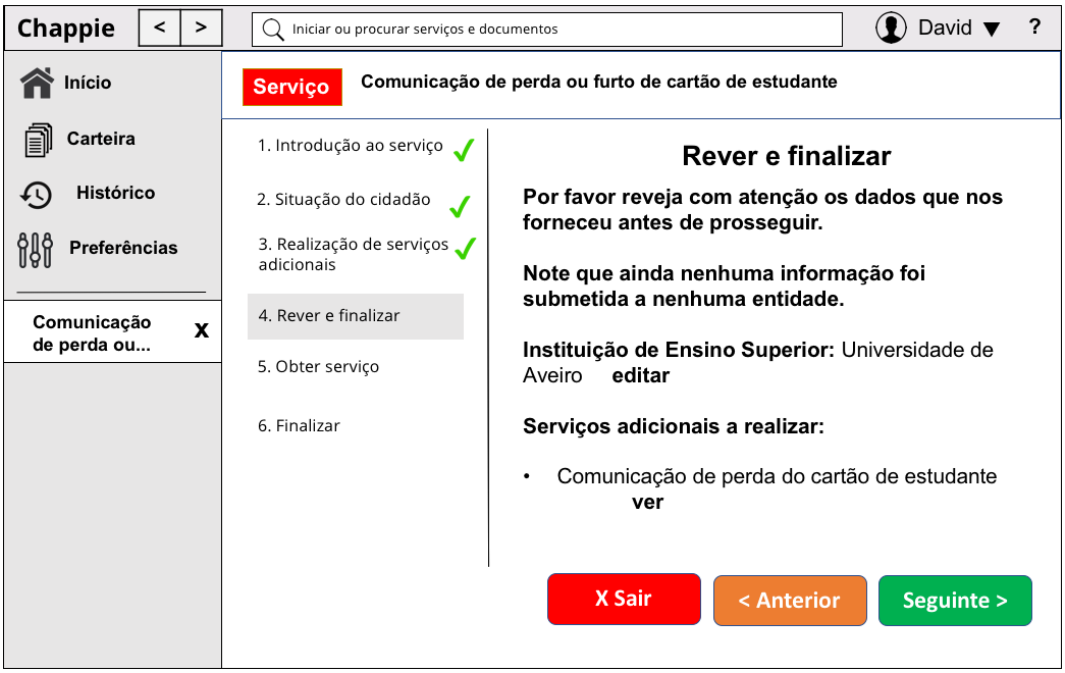

Figure 4.44: Review screen for a service in the second paper prototype

Evaluator ID	Completed the task	Time	Made mistakes	Felt lost	Asked for help
1	Yes	01:00	No	No	No
2	Yes	02:00	No	No	No
3	Yes	01:20	No	No	No
4	Yes	02:50	No	Yes	No
5	Yes	01:50	No	Partially	No
6	Yes	01:00	No	Partially	Yes
7	Yes	02:00	No	No	No
8	Yes	02:10	No	No	No
9	Yes	01:00	No	No	No
10	Yes	02:30	Some	Partially	Yes

Table 4.11: Observer's notes for the third task of the second usability test.

Evaluator ID	Reported ease-of-use	Observed ease-of-use	Observations
1	5	5	-
2	5	4	-
3	5	5	-
4	5	3	-
5	5	4	-
6	5	4	The user did not understand the need for a service
7	4	4	-
8	4	4	-
9	5	5	-
10	5	4	The user did not understand the need for a service

Table 4.12: Reported and observed difficulty of the third task of the second usability test

All the users were able to complete the task, with two one of them (evaluators 6 and 10), asking for help during the process. These two users reported difficulties in understanding the difference between the virtual documents they had to provide in the first task and the process for reporting the lost documents in this task. Both of them were expecting the application to only ask for the information of the first step and proceed accordingly after that without requiring interaction from the user.

4.7.9 Task 4: Obtain the service

Having all the fields filled, it is time for the evaluators to obtain all the documents required for the service and then obtain the service itself.

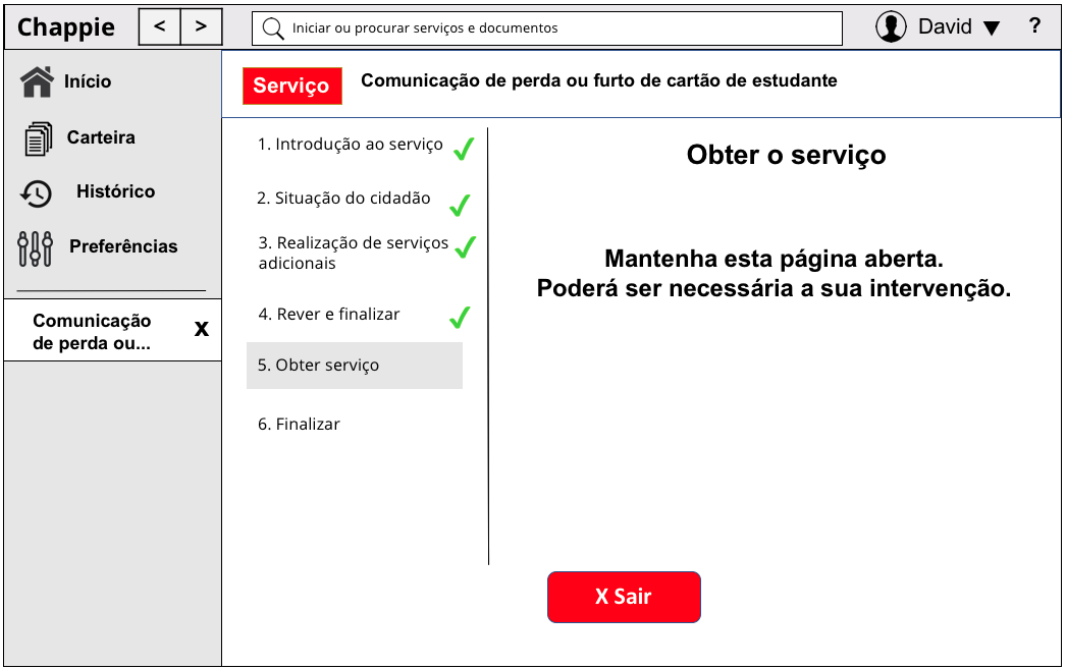

Figure 4.45: Waiting screen shown while obtaining a document

This task serves to understand if it is perceptible that no document was obtained prior to this action.

Expected steps

After clicking the "Next" button shown in figure 4.44, the service starts to be obtained. Users should not leave this page, as their attention may be required for providing additional details. Figure 4.45 shows the screen users are presented that informs them not to leave the page. Figure 4.46 shows the additional solicitation users are required to input values in.

After the service being finished, users are presented with a success message, shown in figure 4.47, where they are able to check new documents that were added to their wallets.

4.7.10 Usability test's results

The results of this task are displayed in tables 4.13 and 4.14. The average time the evaluators took to perform this task was 16,2 seconds, with an highest being of 51 seconds and a lowest of 10 seconds. In average, the evaluators considered the task to be extremely easy, rating them with a difficulty of 5 out of 5 in terms of how easy it is to perform the task.

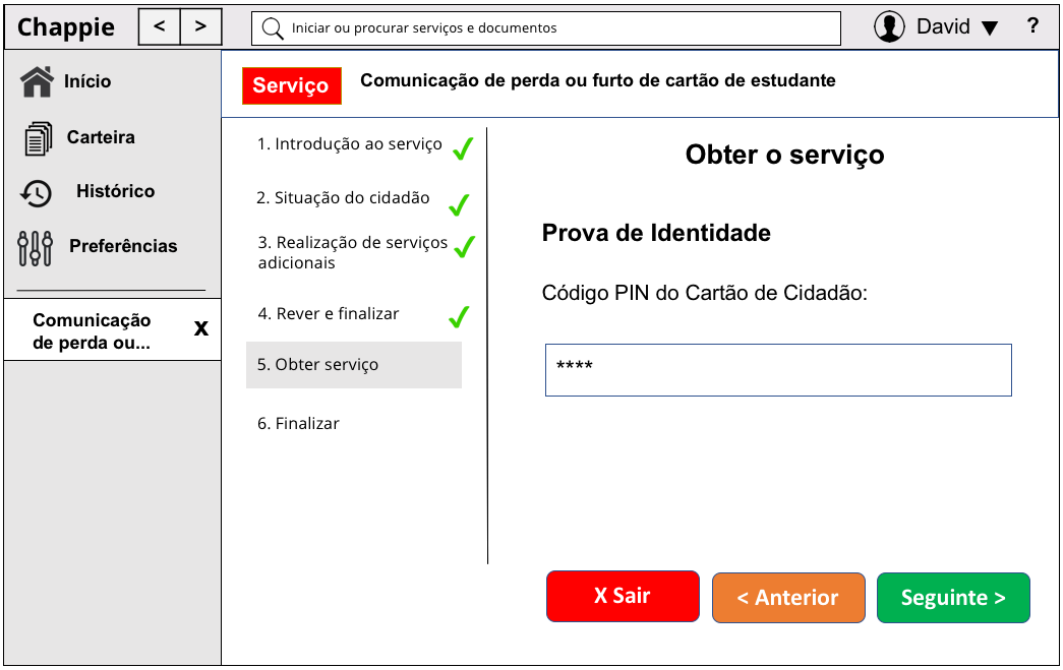

Figure 4.46: Additional solicitation required for service conclusion

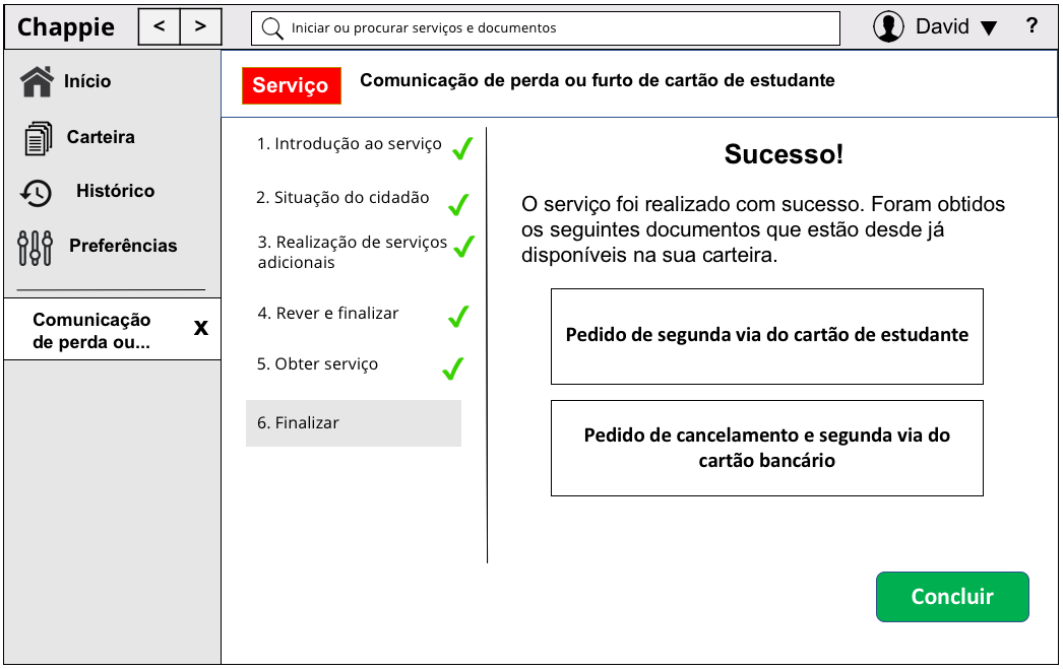

Figure 4.47: Service success screen of the second paper prototype

Evaluator ID	Completed the task	Time	Made mistakes	Felt lost	Asked for help
1	Yes	00:15	No	No	No
2	No	01:00	No	No	No
3	Yes	00:10	No	No	No
4	Yes	00:36	No	No	No
5	Yes	00:51	No	No	No
6	Yes	00:10	No	No	No
7	Yes	00:05	No	No	No
8	Yes	00:10	No	No	No
9	Yes	00:10	No	No	No
10	Yes	00:15	No	No	No

Table 4.13: Observer's notes for the fourth task of the second usability test.

Evaluator ID	Reported ease-of-use	Observed ease-of-use	Observations
1	5	5	-
2	4	5	-
3	5	5	-
4	5	5	-
5	5	5	-
6	5	5	-
7	5	5	-
8	5	5	-
9	5	5	-
10	5	5	-

Table 4.14: Reported and observed difficulty of the fourth task of the second usability test.

The results of these tables shows that this screen met the evaluators' expectations. For the next task, users were asked to access the documents obtained in the service.

4.7.11 Task 5: View obtained documents

This task had the purpose of understanding the preferred menu option for viewing the documents obtained in a service.

Expected steps

After finishing the service, users were returned to the home screen shown in figure 4.33. They were then asked to find the documents they had just obtained.

There were two ways to achieve this goal: through the wallet (Figure 4.48) or using the history feature of the application (Figure 4.49). Technically it should also be possible to find these documents using the search box. However, this last option is a more complex task in terms of the number of required steps and as was outside of the scope of this test.

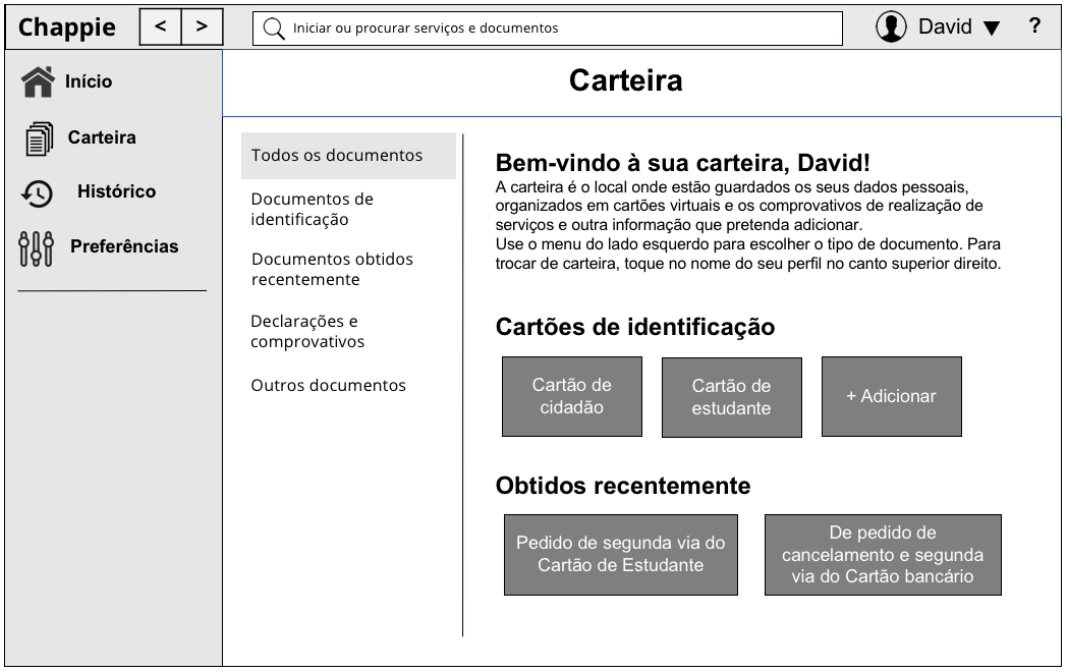

Figure 4.48: Finding obtained documents in the citizen wallet

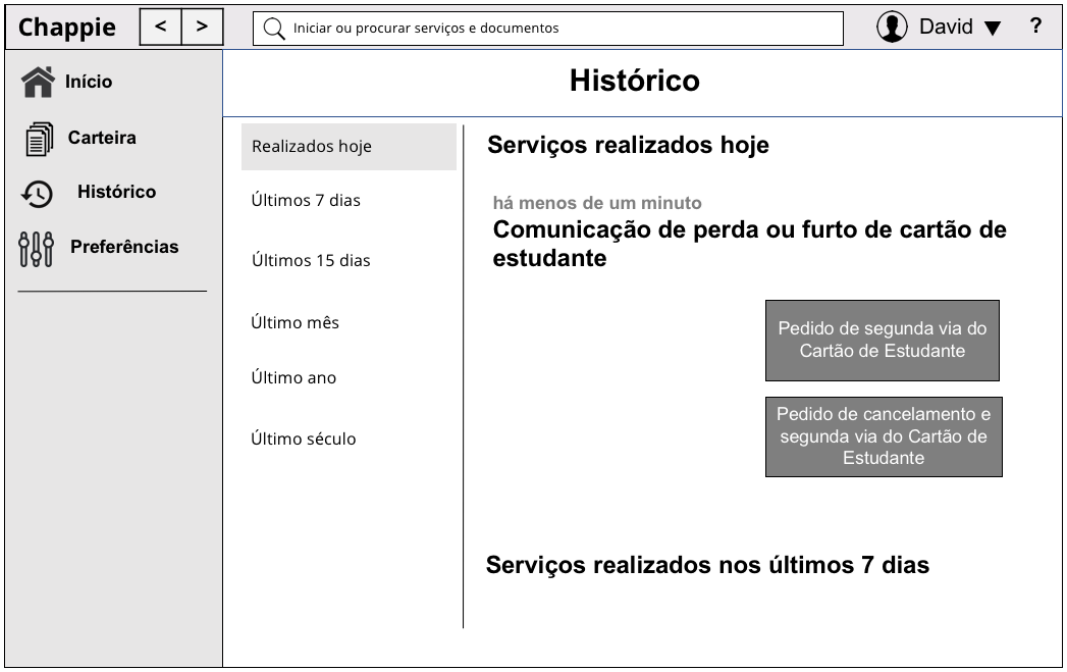

Figure 4.49: Finding obtained documents in the history screen

4.7.12 Usability test's results

As can be seen in table 4.15, the majority of the users selected the "Documents" option when presented with it (7). Evaluator 4 skipped the initial screen with the search box shown in figure 4.33 and only after that selected the "Documents" option. Evaluator 5 tried using the search box shown in that figure and, after being informed that the option was not available in the prototype, selected the "History" option. The same option was also selected by evaluators 7 and 10, with the latter later reporting having had difficulties in finding what option to select.

With the values shown in this table and in table 4.16 it can be concluded that the majority of the users preferred selecting the "Documents" option, but the ability of having an option to view the history of services performed and searching for documents are also features considered important.

Evaluator ID	**Reported ease-of-use**	**Observed ease-of-use**	**Observations**
1	5	5	Selected "Documents"
2	5	3	Selected "Documents"
3	5	5	Selected "Documents"
4	3	3	Skipped Search; Selected "Documents"
5	5	4	Tried using the search box; Selected "History"
6	5	5	Selected "Documents"
7	5	5	Selected "History"
8	4	5	Selected "Documents"
9	5	5	Selected "Documents"
10	5	4	Had difficulties understanding where (s)he had to go; Selected "History"

Table 4.15: Reported and observed difficulty of the fifth task of the second usability test.

4.7.13 Overall appreciation

Evaluators were also asked to classify a set of sentences according to a scale from 1 to 5, where 1 means "Strongly disagree" and 5 means "Strongly agree". The answers given by the evaluators are displayed in table 4.17.

Evaluator ID	Completed the task	Time	Made mistakes	Felt lost	Asked for help
1	Yes	00:20	No	No	No
2	Yes	00:30	Yes	Yes	No
3	Yes	00:10	No	No	No
4	Yes	01:50	No	Partially	No
5	Yes	00:10	Yes	No	Yes
6	Yes	00:20	No	No	No
7	Yes	00:13	No	No	No
8	Yes	00:35	No	No	No
9	Yes	00:40	No	No	No
10	Yes	00:48	No	Partially	No

Table 4.16: Observer's notes for the fifth task of the second usability test.

Statement	1	2	3	4	5	6	7	8	9	10
It is easy to use the system	5	5	5	5	5	4	5	5	5	5
I can easily find what I am looking for	5	5	5	5	5	4	5	4	5	5
The system is slow	3	3	3	1	3	2	1	4	1	4
It is pleasant to use the system	4	4	4	5	4	4	5	5	5	5
The system has some annoying characteristics	4	1	3	1	2	2	1	2	1	2
There is consistency between system elements	4	5	5	5	5	4	5	5	5	5
I need help to use the system	1	1	1	2	1	3	1	1	1	1
Using the system requires prior knowledge of similar applications	2	5	1	1	2	2	1	3	1	1

Table 4.17: Observer's notes for the fifth task of the second usability test.

It was almost consensual that the system is easy and pleasant to use and that it is easy to find information in the system. The majority of the users reported not needing help to use the system. Some evaluators reported that the system has annoying characteristics and prior knowledge of similar applications is required.

Regarding the "The system is slow" statement, all the users asked what the meaning of that sentence was. The sentence refers to the amount of steps required to fulfill a given task in the application. With an average of 2,5 in this criteria and the results shown in table 4.18, we can conclude that users found the amount of steps required to use the application excessive, i.e., there was the need for clicking "Next" an necessary amount of times. One of the evaluators even suggested removing the screens shown when starting each partial service.

Evaluator ID	Feedback
1	System requires too many steps
2	System requires too many steps
3	System requires too many steps
4	-
5	System requires too many steps
6	-
7	-
8	-
9	System requires too many steps
10	Welcome screens in partial services are unnecessary; System requires too many steps

Table 4.18: Feedback given by the users after testing the system

Chapter 5

Developing the new interface

This chapter describes the development process that was carried for the new Chappie User Interface. It covers the technologies that were used for developing the application (section 5.1) and describes the newly developed Chappie API (section 5.2).

The chapter also presents the new Chappie home screen (section 5.3), the initial concept of the dependencies tree building process using a tree view (section 5.4) and the adopted approach for this same process (section 5.5).

Lastly, the chapter covers the libraries that were used for developing the application, where its source code can be found and the instructions for compiling it (section 5.6).

5.1 Usage of Web Technology for the UI

Chapter 3 mentioned some of the problems the old prototype had, one of them being the fact that it is written in Java, a technology that nowadays is not available in most platforms. As a way to guarantee compatibility with the largest number of devices possible, new technology choices had to be made.

The one thing most Operating Systems have in common is a browser for surfing the web. Web applications and websites are particularly interesting because they behave almost the same no matter the platform where they are run, apart from small differences that may exist in different browser rendering engines.

JavaScript is a programming language that was initially created for adding animations and interactive actions to web pages. In more recent years, both the number of users and use-cases have been expanded, making transforming JavaScript in a programming language that can now be used to create complete applications not just for the web, but also for servers, desktops, or mobile phones and other devices like tablets, gaming consoles or Smart TVs.

Because every major platform has a browser or a JavaScript engine nowadays, choosing this language for developing an application makes it possible to run the same code in different platforms with small or no changes being required. Besides, the large community of people using Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) to develop user interfaces makes them good choices for creating an application User Interface.

Of course, running an application directly in a web browser does not allow it to to have access certain features of the system like sensors, storage or network status. These restrictions exist to prevent malicious websites from accessing the user's personal information or performing tasks on the system on his behalf. But that does not mean that those restrictions can not

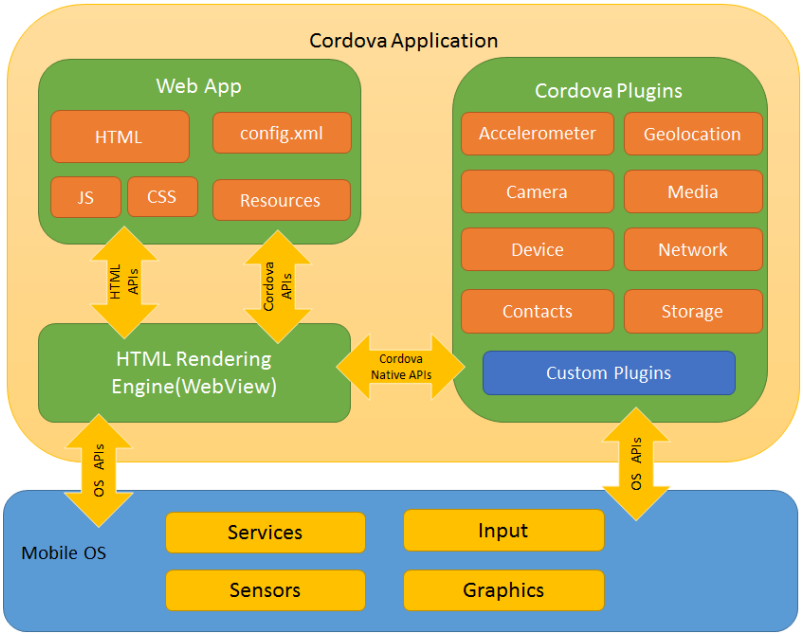

Figure 5.1: High-level view of the Apache Cordova application architecture (source: Apache Cordova Documentation).

---

be bypassed.

Apache Cordova[1] is an open-source mobile development framework that allows the use standard web technologies like HTML5, CSS, and JavaScript for developing cross-platform applications. Applications that use Cordova execute within wrappers that provide API bindings to access the device's capabilities such as sensors, storage, and network status (Figure 5.1).

Apache Cordova is one of the solutions that solves the limitations that were enumerated above. However there is still the need for choosing a suitable application framework for Chappie.

An Application Framework is a set of tools that developers can use to add functionalities to their applications and ease their development. A Framework may include reusable tool sets, application programming interfaces (APIs), user interface libraries (components such as buttons and windows that can be used to create a User Interface), among other features.

Angular is one of the sever JavaScript application frameworks that offer resources for the development of web applications for mobile devices and computers. Angular includes, among other features, Components, Services, and Modules. Components are reusable user interface elements such as buttons, dialog boxes, or input fields, that are written using HTML5, CSS and Typescript[2], a superset of JavaScript that adds static typing to the programing language. Angular Services are instances of application classes that can be shared between components to access specific information required by them, such as the name of the person that is using the application. Angular Modules are a set of components and services that can be reused in other applications.

[1] `https://cordova.apache.org`
[2] `https://www.typescriptlang.org/`

The Ionic[3] Framework is a JavaScript application framework that combines the features of Angular and Apache Cordova to provide Typescript bindings to native device functionalities like notifications and vibration, and components designed to match the design guidelines of the platform where the application is running.

5.2 Chappie API

Although it is technically possible to migrate the old Chappie application completely to JavaScript and take advantage of the features that were mentioned before, doing so would require a significant amount of effort and time.

In order to speed the development of Chappie and take advantage of the work that was already done in the old application, this latter was turned into a server that provides support to a new citizen application using a REST API. This means that Chappie consists in two applications: a citizen application (Chappie Citizen) that runs in the citizen's computer, smart-phone or tablet, and a server application (Chappie Server) running in a computer in the same network where the citizen application is running.

Communication between the Chappie Server and the Chappie Citizen is made using Websockets. WebSockets are a technology that makes it possible to open an interactive communication session between the user's browser and a server[4]. Websockets are bi-directional, meaning both the client and the server are able to send messages to each other, eliminating the need for constantly polling the server for changes.

During the application execution, several messages with commands are exchanged between the Chappie Citizen application and the Chappie Server. Table 5.1 lists the messages that can be sent from the citizen application to the server. Messages sent from Chappie Server to Chappie client are listed in table 5.2.

Command	**Command Description**	**Message Content**
questions	Sent in response to a user for circumstances	An array of questions mapping a circumstance to its respective node in the tree.
treeUpdated	Sent when there was a change in the tree view.	The new tree.
pong	Used to keep the connection alive.	No parameters.

Table 5.2: List of messages sent from the Chappie Server to the Chappie Client

Another change that was made which was necessary to improve the overall experience of the application was changing the process of building the dependencies tree. While the old Chappie prototype required the user to follow a strict order when answering circumstances and even blocked the whole process while the interaction was not finished, the new Chappie Server now allows the citizen to select which nodes he wants to interact with without imposing any particular order. This was achieved with a new algorithm for drawing the dependencies

[3] `https://ionicframework.com/`

[4] "WebSockets" in MDN Web APIs: `https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API`

Command	Command Description	Message Content
beginService	Sent when a new service is about to be started	The URL where Chappie should get the service's RDP.
popProviderQuestions	Get next circumstance from the server	None.
onProviderChosen	Sent when the user selects a service provider	Selected value.
onCustomProviderChosen	Sent when the user selects a provider that was unlisted.	Provider description in JSON format.
obtainService	Start obtaining a service.	Current Service ID.
selectFile	Sent when the user selects one file from his wallet.	The selected file's internal identifier.
setFormValue	Sent after the user inputs form data.	Form and service identification and values.
setSAdicionalValues	Sent after the user inputs data requested in additional solicitation.	Form and service identification and values.
ping	Used to keep the Websockets connection running	No parameters.

Table 5.1: List of messages sent from Chappie Citizen to Chappie Server

tree described in figure 5.2. When compared to the old one, it can be seen that nodes that require interaction from the user are marked and put in a waiting state until the user interacts with them without blocking the whole process of the Life Event.

5.3 Home screen

Before starting obtaining services, the users are shown the screen shown in figure 5.3. From that screens they are able to select the service they want to start or access the other screens of the application. New services can be added to Chappie through the use of a QR Code that describes the service. More information on that code is given in a further section of this document.

5.4 Initial approach to dependencies tree building

In order to address the complaints reported in the second usability test of several steps being required in the wizard, a different approach was considered. This new approach presented the list of required documents in a tree view.

The tree displayed all the relations between services, partial services and documents. The root of the tree was the initial service the citizen was looking for and its leafs are the documents and the partial services that compose those service.

The initial intention was to split the screen vertically in two main parts: a navigation part on the left with the tree, and a details part on the right that would be updated depending on

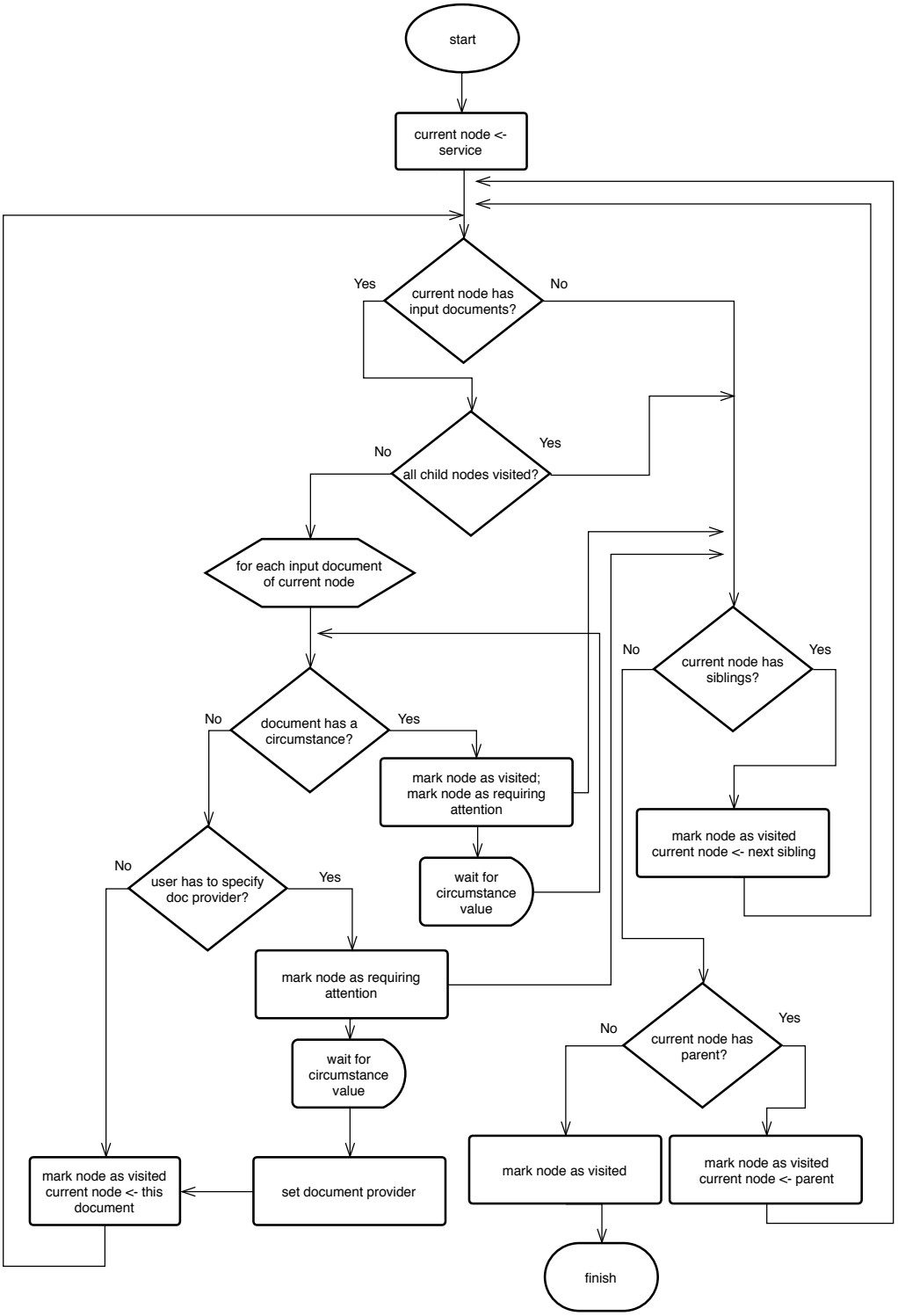

Figure 5.2: Workflow followed by the new Chappie application to create a dependencies tree and obtain all the required documents.

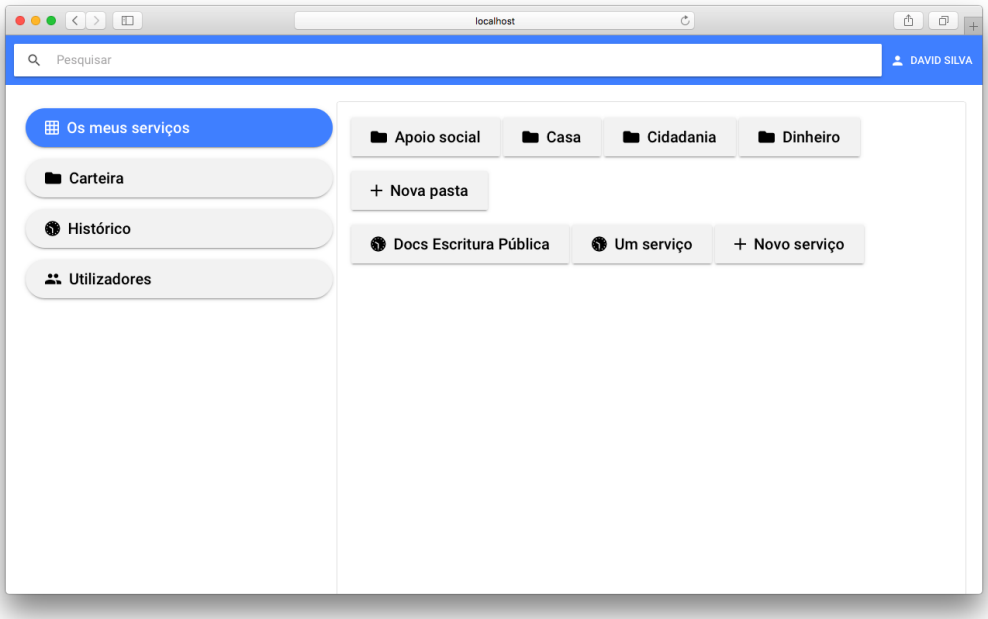

Figure 5.3: The home screen of the final application.

---

the document selected on the left. This concept is shown in figure 5.4, captured during the development of the application. However, using only half of the screen meant there was less horizontal space for the tree to grow.

When a larger testing service was loaded in this view, the resulting experience was very poor because either the tree had to be cut or it would be too far and the user had to zoom in to properly see it. So it was decided that the tree would fill the entire screen.

As a way to decrease the number of touches or clicks from the users, instead of having to go through all the nodes, the ones that required interaction from the user for inputing information or answering a circumstance were marked with a different color, as can be seen in figure 5.5.

The users can select any node from the tree. When the user selects a node, a dialog screen is shown. The displayed screen is organized in three tabs: "Where to Obtain", "Document Attributes" and "Data Input".

The first tab displays the information about the service or document that the selected node represents (figure 5.6). Depending on the node selected, this screen would display the service information or ask the user to answer a circumstance (figure 5.7), upload a document from their devices or select a provider for a service (figure 5.8).

The attributes tab, when available, always displayed the same type of information: the attributes of the documents (figure 5.9).

Some nodes required the users to fill a form. That was the purpose of the third tab: allow the user to fill-in required information (figure 5.10).

Figure 5.11 shows a complete service drawn with this model. As can be seen in the figure, there is a large amount of information being displayed on the screen at the same time.

Two approaches were considered for solving the problem of having too much information

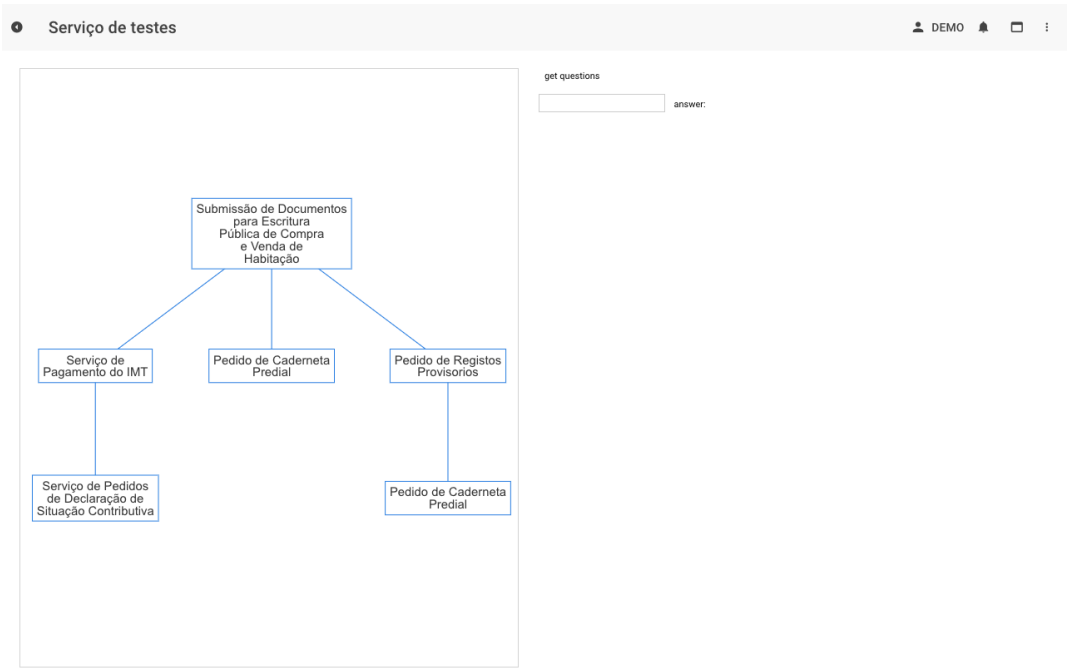

Figure 5.4: Early concept of a tree view using half of the screen

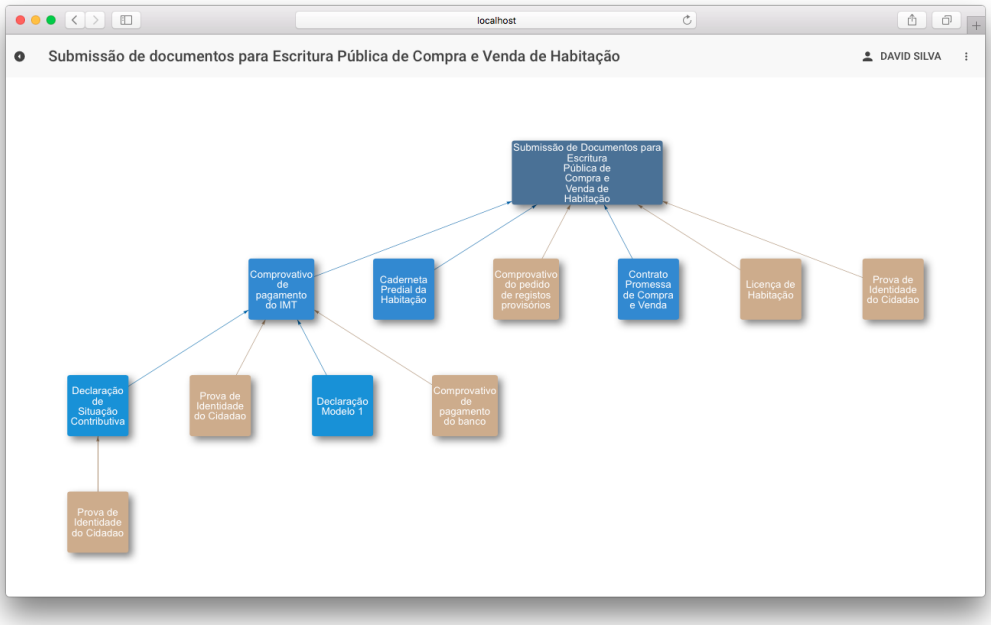

Figure 5.5: An example of nodes requesting attention from the user

Figure 5.6: Service information dialog

Figure 5.7: Service circumstance dialog

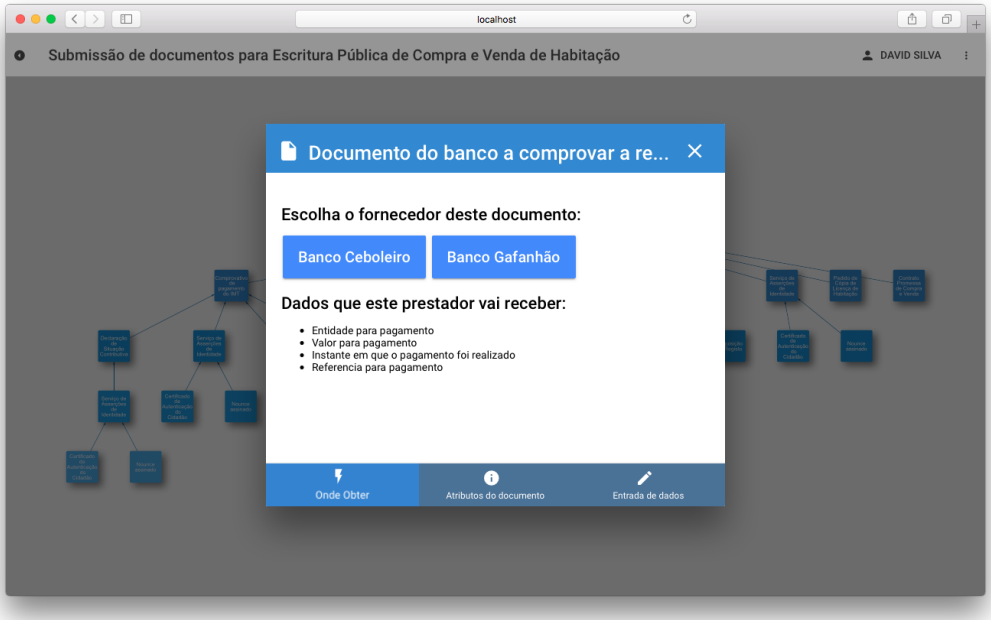

Figure 5.8: Select provider dialog

Figure 5.9: List of document attributes

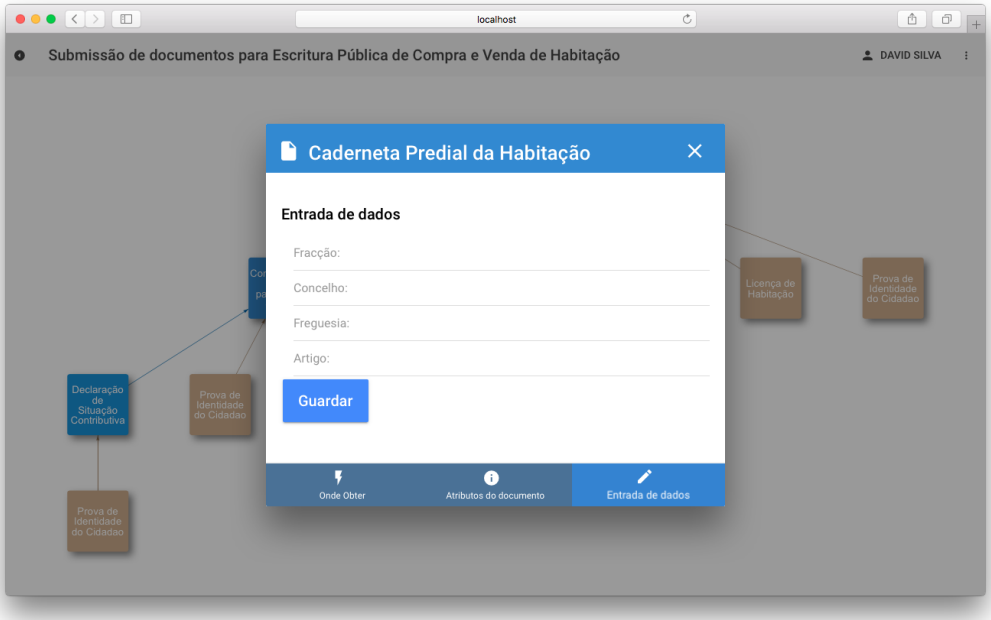

Figure 5.10: Input form dialog

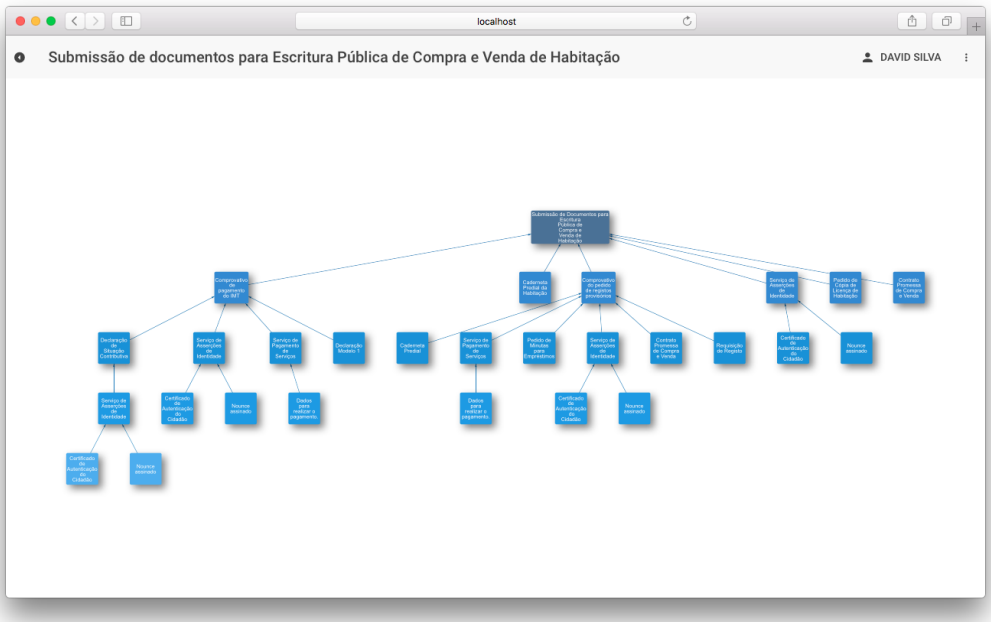

Figure 5.11: The tree drawn using the whole screen.

Age	Sex	Occupation	Preferred view	Reasons
24	M	Student	Wizard	More familiar concept.
24	F	Student	Wizard	Less simultaneous information on screen.
21	F	Student	Wizard	More familiar concept.
23	M	Student	Wizard	More familiar concept.
25	M	Information Security Expert	Wizard	Less simultaneous information on screen
54	F	Waiter	Wizard	Less simultaneous information on screen
25	M	Analyst	Wizard	More familiar concept.
27	M	Project Consultant	Wizard	More familiar concept.

Table 5.3: Success rate of the fourth and fifth task of the first usability test

on the screen at the same time: zooming-in and out automatically on the relevant sections of the interface or draw the tree by levels and focus the interface only in that part.

Changing the nodes to have icons instead of text inside of them was also considered, though this would require every service and document to have its own logo, which in some situations would not make sense and replacing it with a different icon would not add any relevant information.

As such, it was important to understand if it made sense to continue with the development of this user interface or if it should be changed entirely and reverted to the previous approach already studied in usability tests.

5.4.1 Feedback from the users

In order to understand what was the preferred way for people to interact with the system, a group of users was presented with screenshots of both an early concept of the adopted solution and the tree model described above. These users were asked what was their preferred method of visualizing information and the reason why they made that choice. The answers are shown in table 5.3.

All the inquired people preferred the the traditional wizard view. Familiarity was the main reason for their choices, but the information presented in the screen at once also had a significant impact on the preference of the users to the wizard view.

5.5 Adopted approach for obtaining services

The results shown from the feedback of the users clearly state a preference for a more traditional wizard approach. For this reason, and in order to reuse the results obtained from the usability tests, this application was re-engineered to support that approach.

As a way to decrease the number of steps required to obtain a document, the steps shown in the wizard suffered several changes since the initial steps considered in the second usability test.

In the final version, similarly to what happened in the initial prototype, each service or document is represented as a single item in the navigation tree of the wizard. In case the

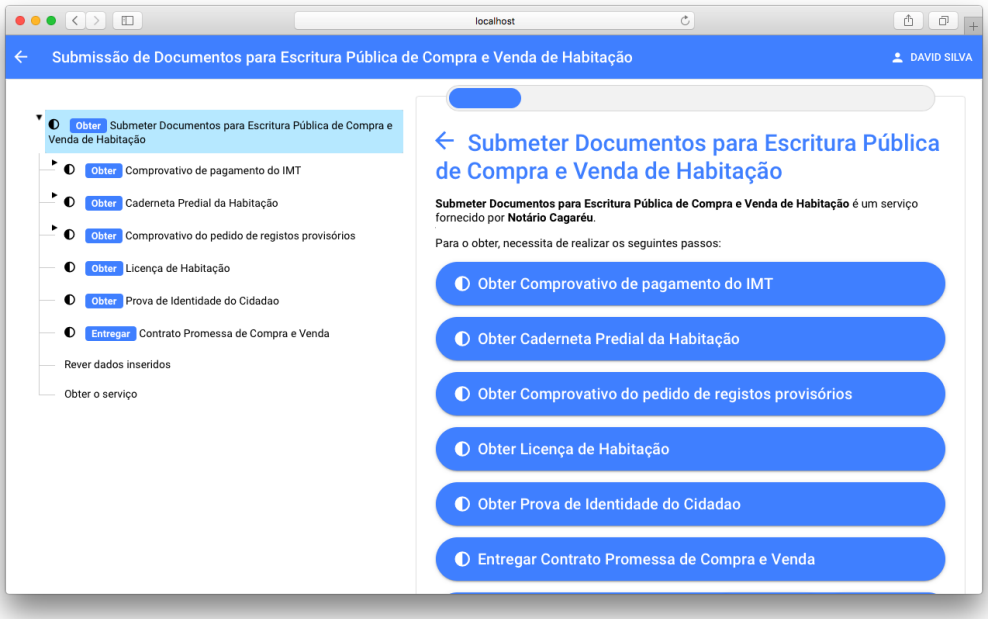

Figure 5.12: Welcome screen on the final application

service requires interaction from the user, being it a circumstance, input, or the need for a document, these required actions are added as child items of the service, as can be seen in figure 5.12.

This change removed the need for clicking "Next" in an initial welcome screen for the service like the ones shown in the first and second paper prototypes, and also increases legibility of the required actions.

Another change that was made was the addition of a verb describing the action related to the service or document in question. Whenever an item in the tree represents a service, it is preceded with the word "Obter" (Obtain) followed by the name of the document the service requires (eg.: "Obter Comprovativo de pagamento do banco" (Obtain proof of payment from the bank)). Documents that have to be provided by the citizen are now preceded by the word "Entregar" (Deliver) (eg.: "Entregar Declaração Modelo 1" (Deliver Model 1 Declaration). This makes the citizen more aware of the type of document that a given item in the navigation tree refers to.

Regarding circumstances (Figure 5.13) and input of information (Figure 5.14), those items are now marked as "Indicar" (Indicate) and "Circunstância" (Circumstance) in the navigation tree. This also allows the user to easily distinguish these items from other items that represent documents or services.

When viewing a partial service, the user is able to see the attributes that are present in the documents issued by that service by clicking the button "Ver atributos do documento" (See document attributes) (Figure 5.15). This opens a pop-up shown in figure 5.16 where the attributes are listed. That list also describes if the documents are used for aggregation (i.e., used to identify the citizen in different documents) and if their value is obfuscated (i.e., the documents are required to identify the citizen but their value is encrypted).

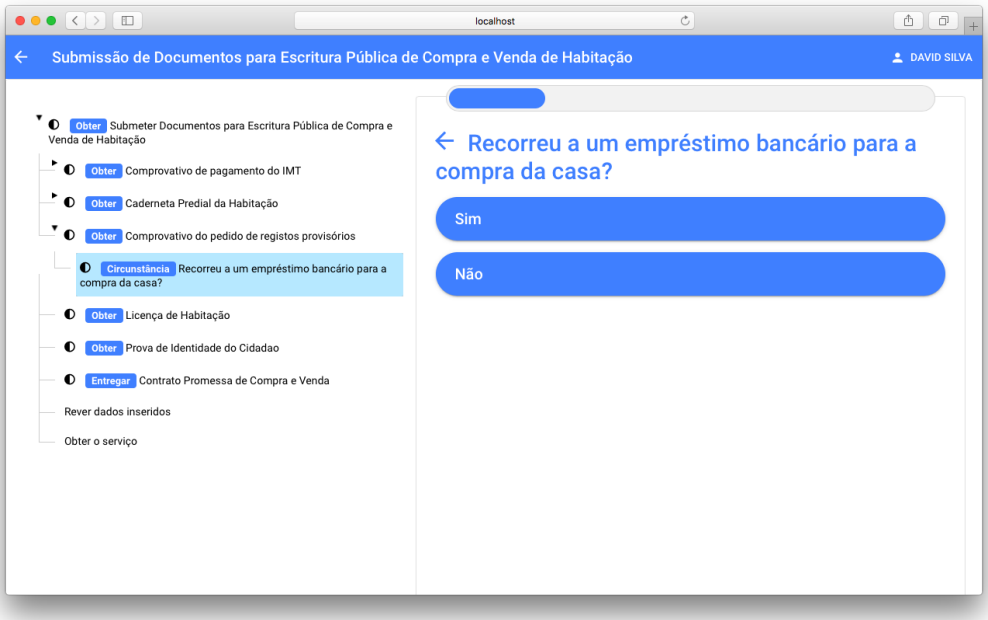

Figure 5.13: Circumstance screen in the final application

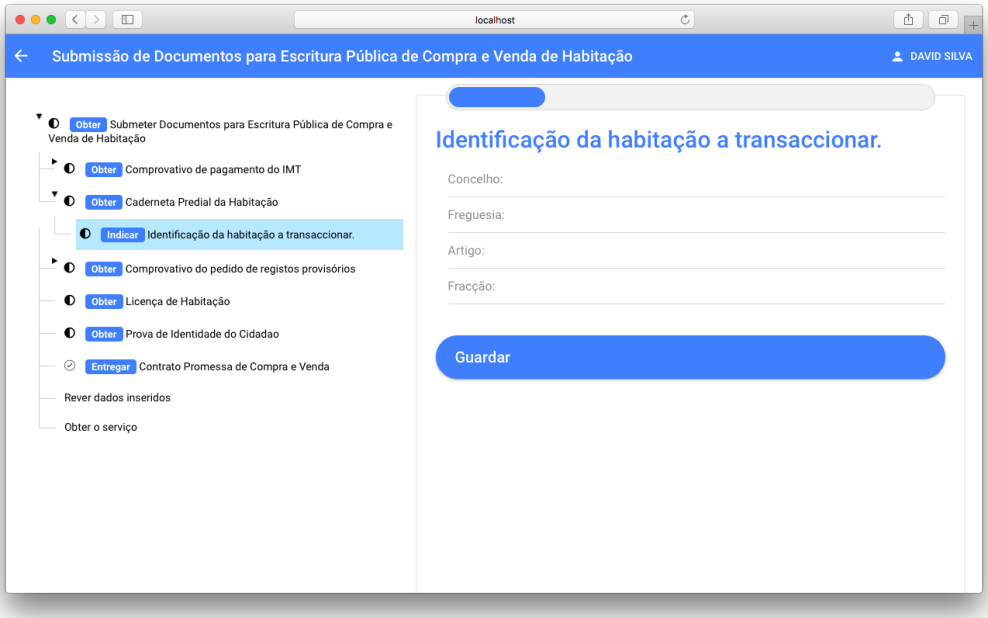

Figure 5.14: Input form in the final application

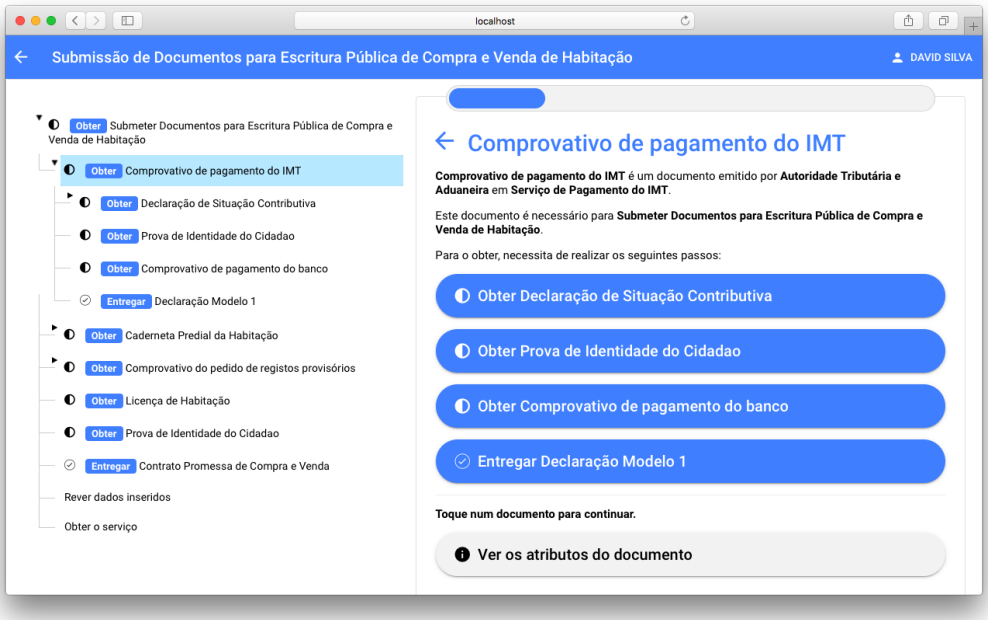

Figure 5.15: List of required documents for a partial service

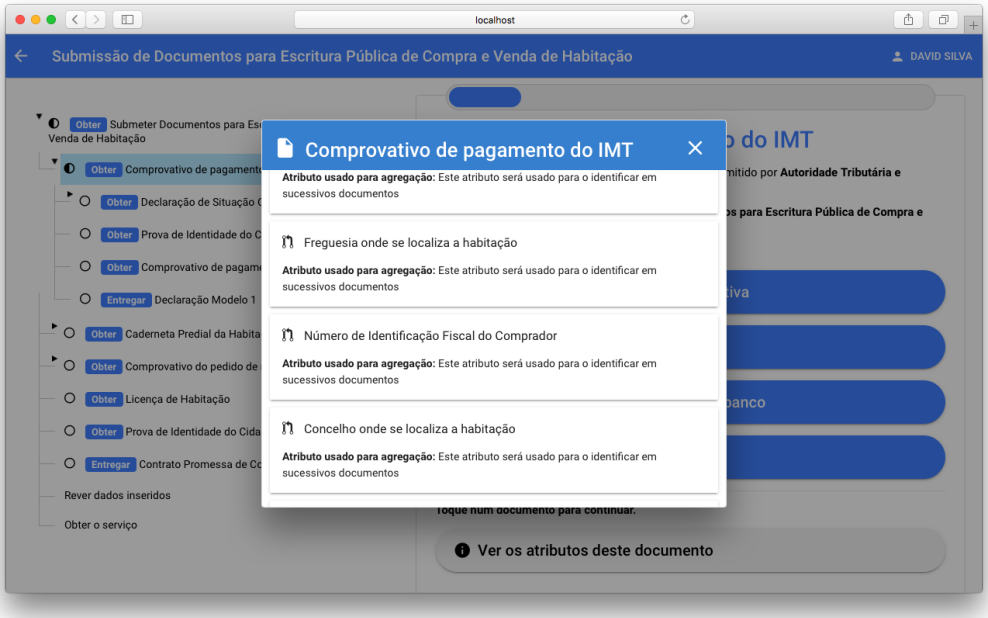

Figure 5.16: List of document attributes

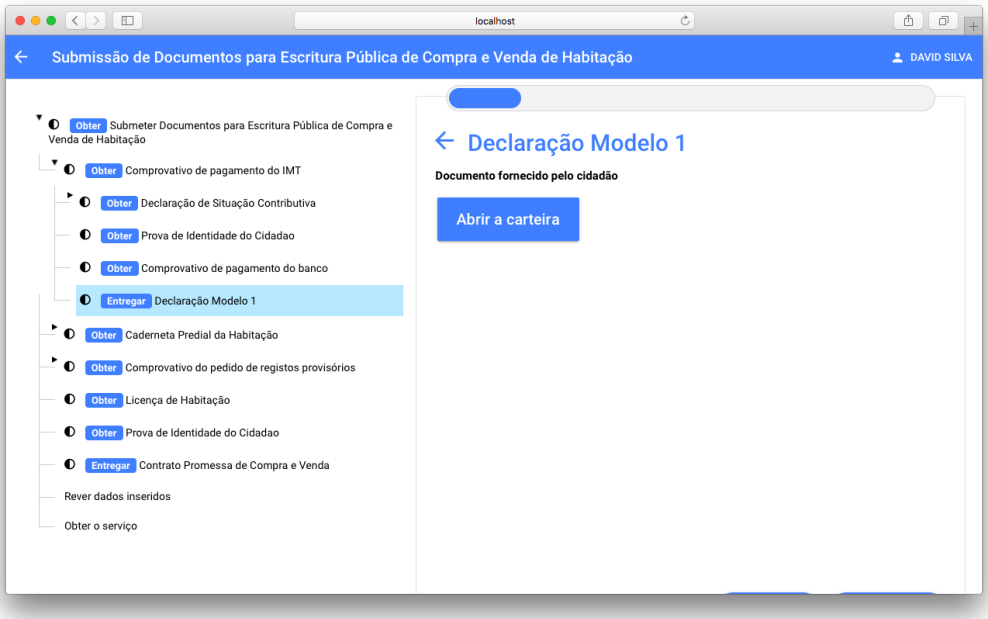

Figure 5.17: Document that the citizen must upload

Delivering a document required for a service is a two-step process: the first one is selecting the navigation item related to that document (Figure 5.17) and the second is select the document from the citizen's wallet (figure 5.18).

Regarding situations where more than one provider can provide a given document, users are now able not only to select a provider from a list of known providers, but also have the option to use their device's camera to scan for a QR Code with the provider information (Figure 5.19).

Another aspect that was included in this version was the status of obtaining each document: items have different iconography displayed next to their name:

- An open circle is displayed when items have to be visited by the user;
- A semi-filled circle means that the user has visited the item but has not yet provided the information the item is requesting;
- A circle with a check means that the item does not require interaction or the user has already interacted with it;
- A filled circle with a check means that the document was already obtained.

Besides the visual indicators for each navigation item, the service also displays a progress bar on the top that indicates the current process of obtaining the service.

At any time, the user is able to review the list of documents or services and their status by clicking the "Review" option on the left 5.20. This is also the screen to where the user is automatically transited to when all the requirements for the service are fulfilled. From here,

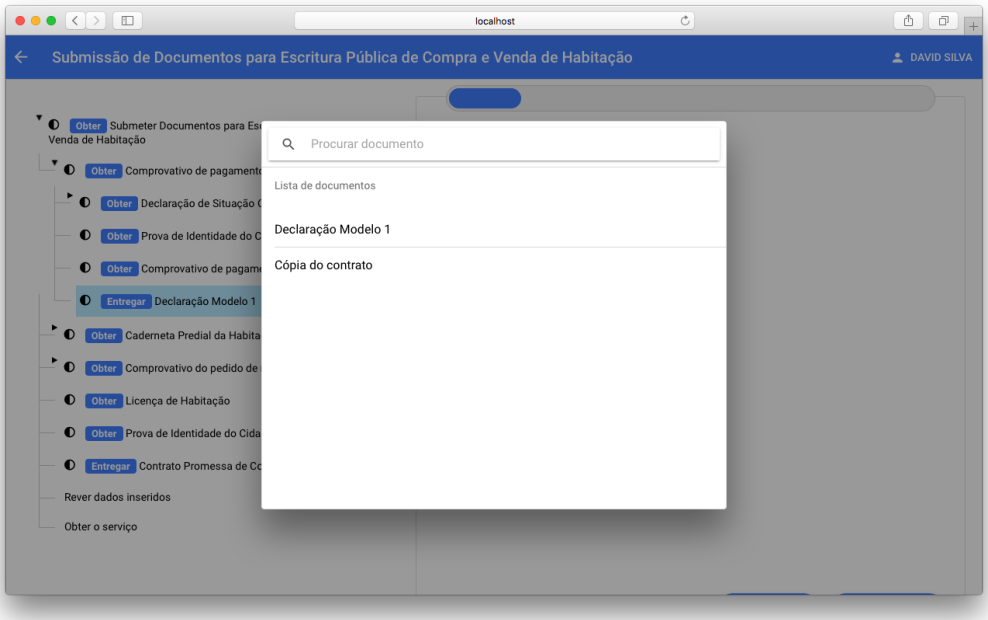

Figure 5.18: Citizen wallet for selecting the document to upload

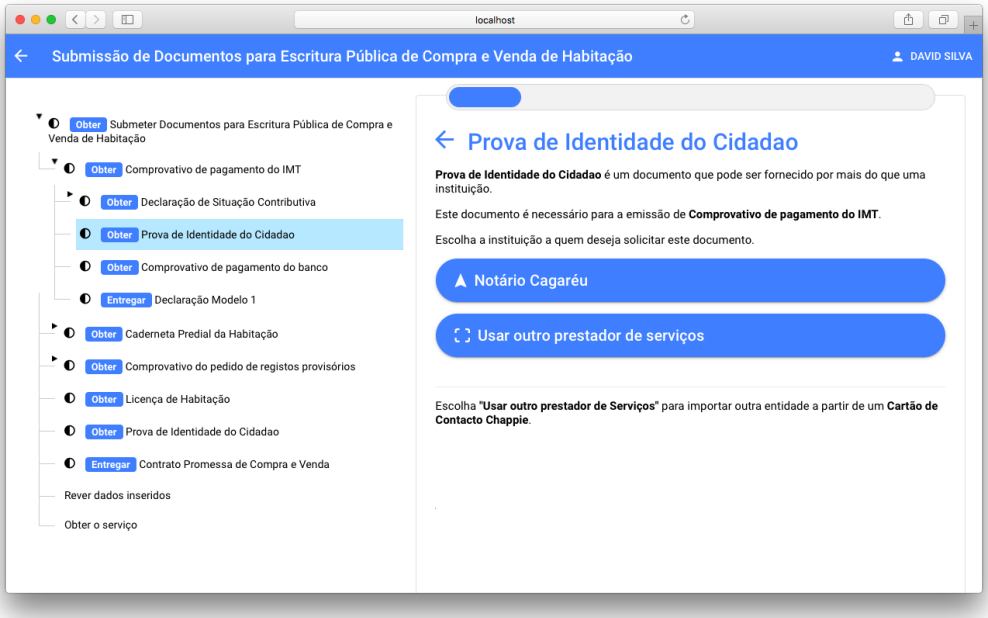

Figure 5.19: Select provider screen in the final application

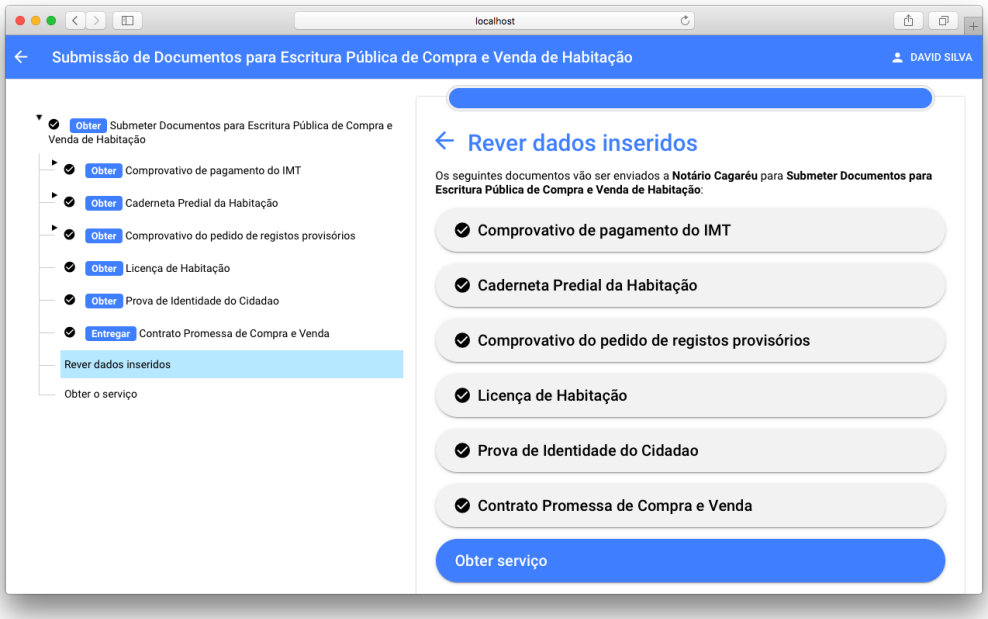

Figure 5.20: Review screen in the final application

the user can navigate to each individual service or document and read its details or proceed obtaining the service by clicking the "Obter serviço" (Obtain service) button on the bottom.

During the obtaining stage, the user may be shown different additional solicitations like the ones seen in the paper prototypes. After the service is finished, the user is greeted with a congratulatory message and can then safely leave the service and return to the home screen (figure 5.21).

5.6 Technical details of the implemented solution

Besides the use of Ionic as the framework for drawing the user interface and its plug-ins combined with Cordova for accessing the device's native functionalities, two other libraries were used to draw the user interfaces: visjs[5] and Angular Tree Component[6]. The first was used to draw the graph user interface shown in the first approach and the second is used to draw the tree view shown in the second approach.

It was mentioned above that Chappie can scan QR Codes. However, we did not specify the contents that the QR Codes must have. The expected value for those codes are JSON-encoded values with the following specification:

```
{
    "version": "1.0",
    "type": "chappie-service",
```

[5] http://visjs.org/
[6] https://angular2-tree.readme.io/

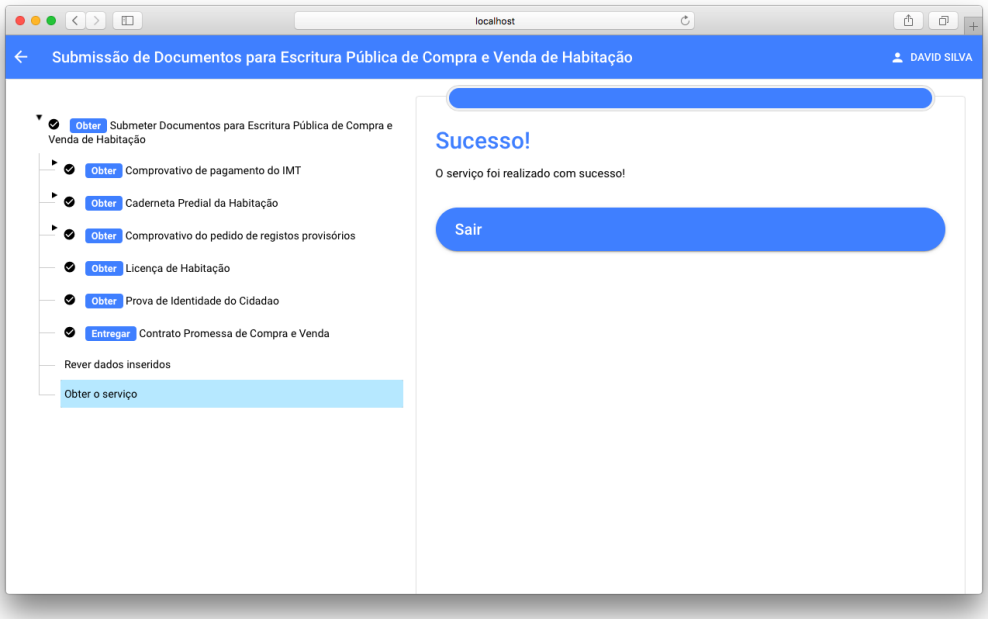

Figure 5.21: Success screen in the final application

```
    "payload": {
        "address": "URL of the root service (eg.: http://www.ua.pt/)",
        "serviceName": "The name of the service",
        "providerName": "Name of the provider",
        "sectorName": "Provider sector (eg.: Bank) (optional)"
    }
}
```

The scanned QR Codes can be used to add new providers to the citizen wallet, new services to the home screen, or to select a custom provider.

Running the application in the development environment requires yarn[7] or npm[8] and cloning both the citizen and server applications from their respective repositories in Code UA[9][10].

The application must first be installed using the `npm install` or `yarn install` command and only then can be executed with the usage of the `ionic run` command.

When running in development mode, it is not possible to read QR Codes or take advantage of the Cordova functionalities. In order to enable these functionalities, the application must be compiled in production mode, which can be done using the `ionic cordova run` command followed by the target platform (`browser`, `ios`, or `android`). Running the application in production mode requires having the required SDKs installed in the machine.

[7] `https://yarnpkg.com/`
[8] `https://www.npmjs.com/`
[9] `http://code.ua.pt/projects/chappie-citizen`
[10] `http://code.ua.pt/projects/chappie-server`

Chapter 6

Conclusion

In this chapter we present a consideration on the work that was done (section 6.1) and present a list of proposals for future work (section 6.2).

6.1 Considerations on the work done

In this work we made two usability tests to find a suitable User Interface for the Chappie application. As a result from those studies, two applications were developed: one showing the dependencies between services structured in the form of a tree (achieved using a graphs library) and another approach that followed a more traditional concept (wizards), where users were guided in the process of obtaining a service using the CHAPAS model.

Based on the feedback obtained from a group of users in the initial stages of development of both applications, the second approach was the one chosen to be presented as the proposed solution for this work. The reasons invoked by these users were the familiarity of concepts and the reduced amount of information being displayed on the screen at the same time.

Several iterations of development were made to achieve the minimum amount of steps required to performing a service in the CHAPAS model without making the user feel lost while performing such action. However, because no additional usability studies were conducted after the application was developed, some important aspects may have been skipped.

Apart of the development of the User Interface itself, this work also resulted in the split of Chappie in two applications: the citizen application described above and a server application that assists it using an API and a WebSockets connection.

Regarding the technological choices made, the solution proposed in this work was based on Cordova but there are other technologies that achieve similar results with different technologies and implementations, and, sometimes, even better performance. Examples of other tools are React Native[1], Microsoft Xamarin[2] or NativeScript[3]. However, at the time of writing, these solutions could not deploy applications that were simultaneously compatible with both mobile devices and computers.

[1] `https://facebook.github.io/react-native/`
[2] `https://visualstudio.microsoft.com/xamarin/`
[3] `https://www.nativescript.org/`

6.2 Future work

This work presents one possible vision of the Chappie application, but there are still some improvements that can be done to improve the application. For starters, conducting a usability test on the purposed application is a good way to detect potential situations that may have been missed during the development process.

Another interesting idea would be to allow more than one service to be run at the same time or allow the application to run in the background and take advantage of the notifications system of the citizen's device for when interactions are required. This is specially important in services where the generation of the documents are not immediate, which was a situation that was not explored in this work.

Because the CHAPAS Model is fully decentralized and there is not a central institution in control of the full life-event service, a more in-depth study has to be made regarding handling and recovering of failures, especially in situations that are related with the rollback of services already obtained and that may involve the devolution of money. This is a feature not implemented in the old prototype and, for that reason, also not considered here, but in a usability perspective is an extremely important issue that needs to be addressed.

Additionally, because the same partial service can be requested multiple times during service provision for providing the same document to more than one institution, a nice-to-have feature would be allowing the reusage of documents. This concept was explored in the paper prototypes and was well received by the users. However, due to the need for additional changes both to the Chappie Server application and the CHAPAS Model, it was not implemented in this work.

Lastly, for those who are interested in multi-modal interaction, creating an interface based on voice interaction would be an interesting concept. This was initially proposed in the early stages of this work, as can be seen in both figures 4.1 and 4.2 (there is a button that reads "Modo guiado" (Guided mode) that represents an early idea for voice controls with text-to-speech feedback).

It should be noticed, however, that trying to make Chappie a virtual assistant could result in a very poor user experience as deciding the number of steps a user must take to obtain a service in Chappie is challenging, specially for voice interactions. If they were two little, the resulting experience would be very similar to the one of the old prototype and thus suffer the same usability issues it already had. However, if the approach followed was very descriptive, the user could easily get lost in the service.

List of References

[1] International Telecommunications Union, "ICT facts and figures 2017," Itu, pp. 1–8, 2017, accessed: 2018-02-05. [Online]. Available: http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2017.pdf

[2] European Comission, "Buying online and solving disputes online: 24.000 consumers used new european platform in first year," 2017, accessed: 2018-02-05. [Online]. Available: http://europa.eu/rapid/press-release_IP-17-727_en.pdf

[3] D. Tinholt, N. van der Linden, A. Groeneveld, E. Sem, R. Cosmina, G. Cattaneo, and S. Aguzzi, eGovernment Benchmark 2017, 2017, vol. 1.

[4] T. Christensen and P. Lægreid, "The whole-of-government approach to public sector reform," Public Administration Review, vol. 67, no. 6, pp. 1059–1066. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6210.2007.00797.x

[5] United Nations, e-Government Survey 2008: From e-Government to Connected Governance, 2008.

[6] M. A. Wimmer and E. Tambouris, Online One-Stop Government. Boston, MA: Springer US, 2002, pp. 117–130. [Online]. Available: https://doi.org/10.1007/978-0-387-35604-4_9

[7] D. Cukjati and L. Todorovski, "Integrating E-Services in Public Administration: Analysis of EU Research Projects and Involvement of Participants From See Region," Architecture, 2008.

[8] J. S. H. France Belanger, "A framework for e-government: privacy implications," in Business Process Management Journal, Vol. 12 Issue: 1, 2002, pp. 48–60.

[9] H. Gomes, "Serviços orientados a eventos da vida controlados pelo cidadão," Ph.D. dissertation, Universidade de Aveiro, 2015.

[10] H. C. Relyea, "E-gov: Introduction and overview," Government Information Quarterly, vol. 19, no. 1, pp. 9–35, 2002.

[11] M. Yildiz, "E-government research: Reviewing the literature, limitations, and ways forward," Government Information Quarterly, vol. 24, no. 3, pp. 646–665, 2007.

[12] V. Ndou, "E-government for developing countries: Opportunities and challenges," The Electronic Journal of Information Systems in Developing Countries, vol. 18, 2004.

[13] F. D. Brí and F. Bannister, "Whole-of-government: The continuing problem of eliminating silos," in Proceedings of the 10th European Conference on eGovernment. National Centre for Taxation Studies and University of Limerick, 2010, pp. 122–133.

[14] E-Government Survey 2010: Leveraging E-government at a Time of Financial and Economic Crisis. United Nations, 2010.

[15] M. Vintar, M. Kunstelj, and A. Leben, "Delivering better quality public services through life-event portals," 10th NISPAcce Annual Conference, Cracow, Poland, 2002.

[16] G. P. Dias, "Q-model: Um modelo bidimensional de maturidade para o e-government," RISTI: Iberian Journal on Information Systems & Technologies, vol. 7, 2011.

[17] G. P. Dias and J. A. Rafael, "A simple model and a distributed architecture for realizing one-stop e-government," Electronic Commerce Research and Applications, vol. 6, no. 1, pp. 81–90, 2007.

[18] G. P. Dias and T. Narciso, "Analysis of the potential for organizational interoperability improvement in local government," in Actas de la 5ª Conferencia Ibérica de Sistemas y Tecnologias de Información, 2010, pp. 167–172.

[19] F. Bannister, "The panoptic state: Privacy, surveillance and the balance of risk," Information Polity - Public Administration in the Information Society: Essays in Risk and Trust, vol. 10, no. 1, 2, pp. 65–78, 2005.

[20] C. Bellamy, "Joining-up government in the uk: Towards public services for an information age," Australian Journal of Public Administration, vol. 58, no. 3, 2002.

[21] I. Kushchu and M. H. Kuscu, "From e-government to m-government: Facing the inevitable."

[22] I. Trochidis, E. Tambouris, E. Tambouris, and K. Tarabanis, "Identifying common workflow patterns in life-events and business episodes," in The Second International Conference on e-Government, 2006, pp. 234–243.

[23] A. Leben and M. Bohane, Architecture of an Active Life-Event Portal: A Knowledge-Based Approach. IFIP International Federation for Information Processing, 2004.

[24] D. Pappa and C. Makropoulos, "Designing a brokerage platform for the delivery of e-government services to the public," in Knowledge Management in Electronic Government, M. A. Wimmer, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 178–189.

[25] "Wizards," https://docs.microsoft.com/pt-pt/windows/desktop/uxguide/win-wizards, accessed: 2018-02-07.

[26] "Gartner says by 2019, 20 percent of user interactions with smartphones will take place via vpas," http://www.gartner.com/newsroom/id/3551217, accessed: 2017-06-03.

[27] "Peoplesoft benefits life events," https://www.youtube.com/watch?v=5a2q3hVqQbg, accessed: 2017-06-03.

[28] N. Bevan, "Human-computer interaction standards," in Symbiosis of Human and Artifact, ser. Advances in Human Factors/Ergonomics, Y. Anzai, K. Ogawa, and H. Mori, Eds. Elsevier, 1995, vol. 20, pp. 885 – 890.

[29] M. O. Leavitt and B. Shneiderman, Research-Based Web Design & Usability Guidelines. United States Government Printing Office, 2006.

[30] N. Bevan and L. Spinhof, Are guidelines and standards for web usability comprehensive? Springer, 2007.

[31] J. Nielsen, "10 usability heuristics for user interface design," https://www.nngroup.com/articles/ten-usability-heuristics/, 1995, accessed: 2018-02-05.

[32] C. Wharton, J. Rieman, C. Lewis, and P. Polson, "Usability inspection methods," J. Nielsen and R. L. Mack, Eds. New York, NY, USA: John Wiley & Sons, Inc., 1994, ch. The Cognitive Walkthrough Method: A Practitioner's Guide, pp. 105–140.

[33] L. Todorovski, A. Leben, M. Kunstelj, D. Cukjati, and M. Vintar, "Methodology for building models of life events for active portals," EGOV 2006, 2006.